CSE 105:
Data Structures and Algorithms-I
(Part 2)

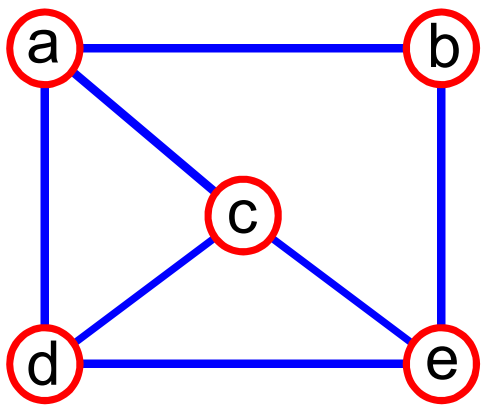
Instructor

Dr Md Monirul Islam

Graphs and Trees: Representation and Search

Graph: Definition

- A graph is a pair (V, E), where
  - V is a set of nodes, called vertices
  - E is a collection of pairs of vertices, called edges
- V(G) and E(G) represent the sets of vertices and edges of G, respectively
- Example:

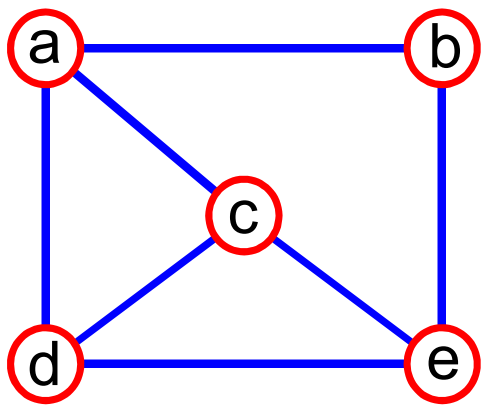


$V = \{a, b, c, d, e\}$

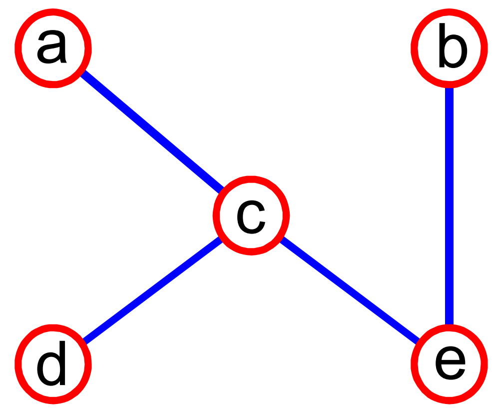
$E = \{(a,b), (a,c), (a,d), (b,e), (c,d), (c,e), (d,e)\}$

Graph: Definition

- A tree is a special type of graph!
- A tree is a graph that is connected and acyclic.



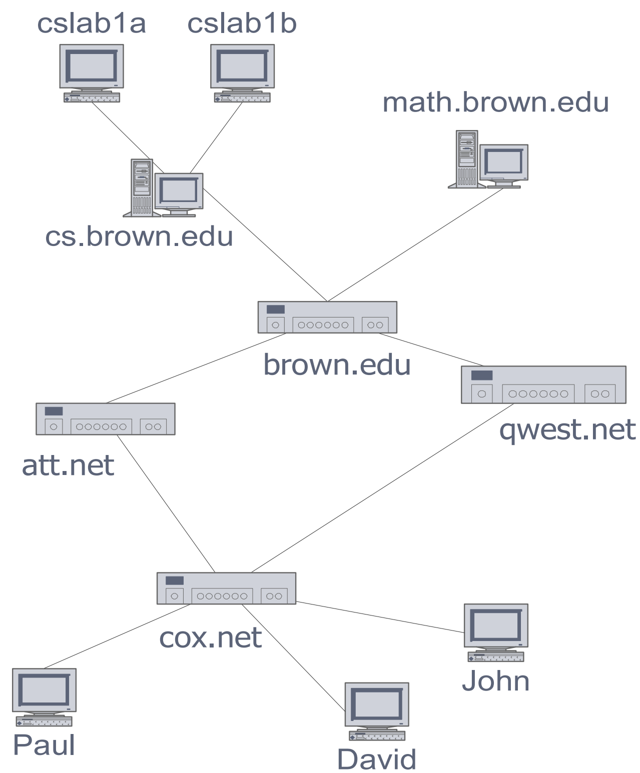
Graph



Tree

Applications

- **Electronic circuits**
  - Printed circuit board
  - Integrated circuit
- **Transportation networks**
  - Highway network
  - Flight network
- **Computer networks**
  - Local area network
  - Internet
  - Web
- **Databases**
  - Entity-relationship diagram



What can we do with graphs?

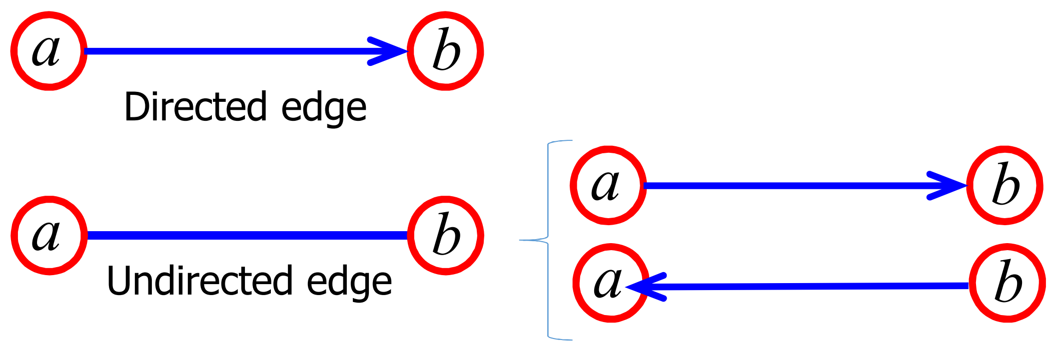
- Find a *path* from one place to another
- Find the *shortest path* from one place to another
- Determine connectivity
- Find the "weakest link" (min cut)
    - check amount of redundancy in case of failures
- Find the amount of flow that will go through them

Edge and Graph Types

- Directed edge
  - ordered pair of vertices (***a***, ***b***)
  - first vertex ***a*** is the origin
  - second vertex ***b*** is the destination

Edge and Graph Types

- **Directed edge**
  - ordered pair of vertices (***a***, ***b***)
  - first vertex ***a*** is the origin
  - second vertex ***b*** is the destination
- **Undirected edge**
  - unordered pair of vertices (***a***, ***b***)

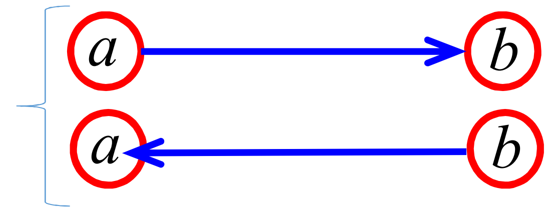


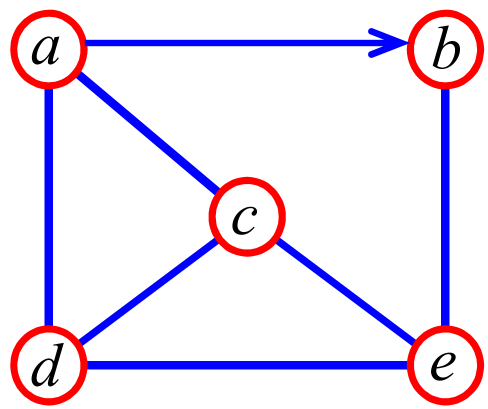
Edge and Graph Types

- **Directed edge**
  - ordered pair of vertices (***a***, ***b***)
  - first vertex ***a*** is the origin
  - second vertex ***b*** is the destination
- **Undirected edge**
  - unordered pair of vertices (***a***, ***b***)
- **Directed graph (Digraph)**
  - all the edges are directed
- **Undirected graph**
  - all the edges are undirected
- **Mixed graph**
  - some edges are undirected and some edges are directed

Directed edge

Undirected edge

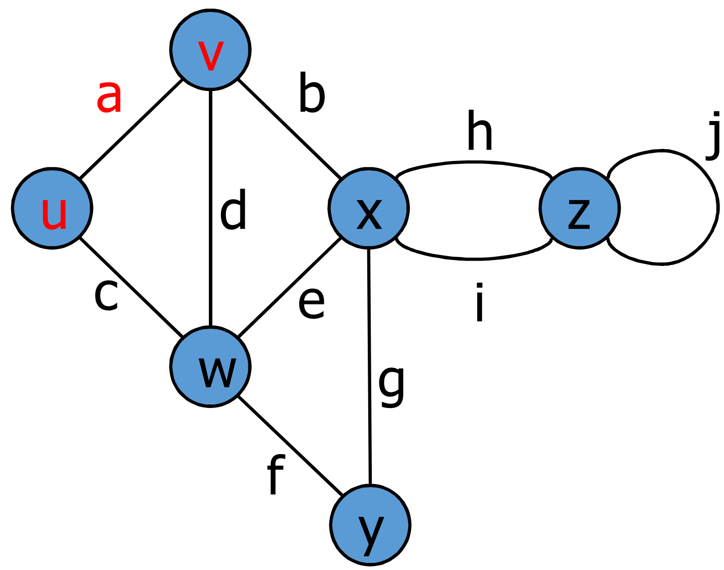




Mixed graph

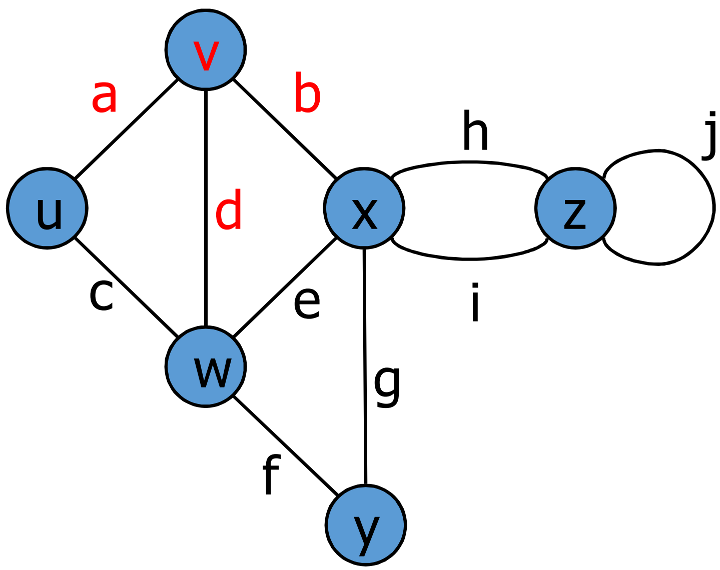
Terminology

- End vertices (or endpoints) of an edge
  - *u* and *v* are the *endpoints* of *a*



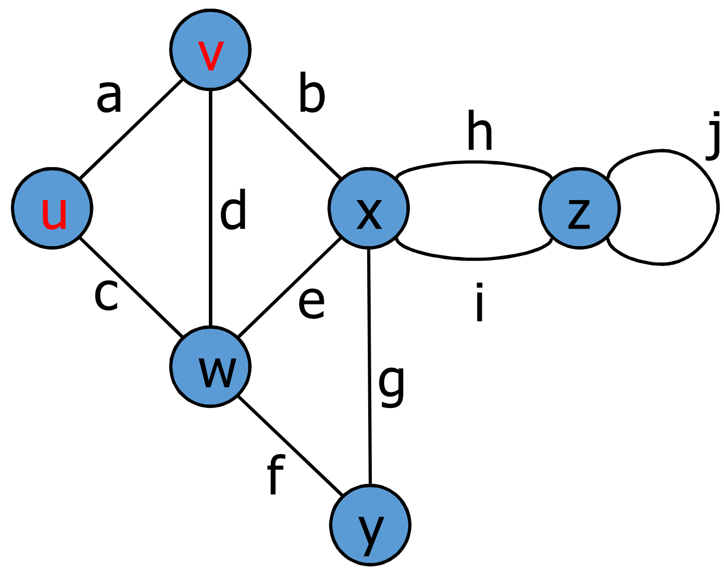
Terminology

- End vertices (or endpoints) of an edge
  - *u* and *v* are the *endpoints* of *a*
- Edges incident to a vertex
  - *a, d*, and *b* are *incident* to *v*



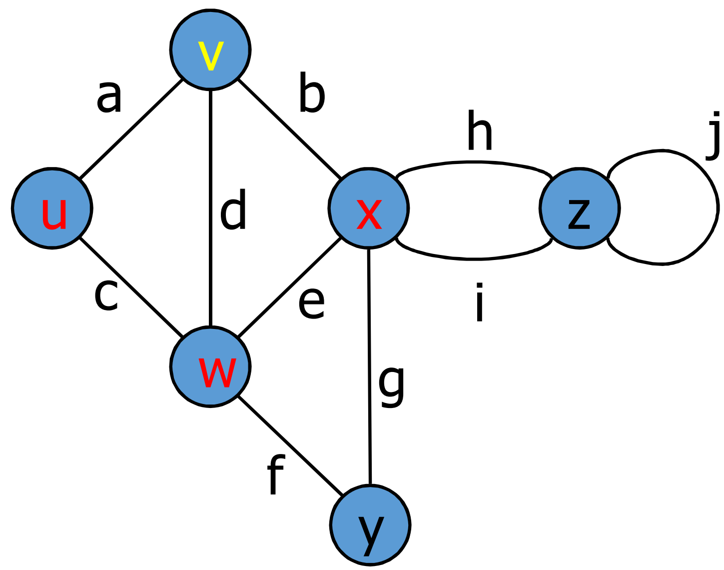
Terminology

- End vertices (or endpoints) of an edge
  - *u* and *v* are the *endpoints* of *a*
- Edges incident to a vertex
  - *a, d*, and *b* are *incident* to *v*
- Adjacent vertices
  - *u* and *v* are *adjacent*



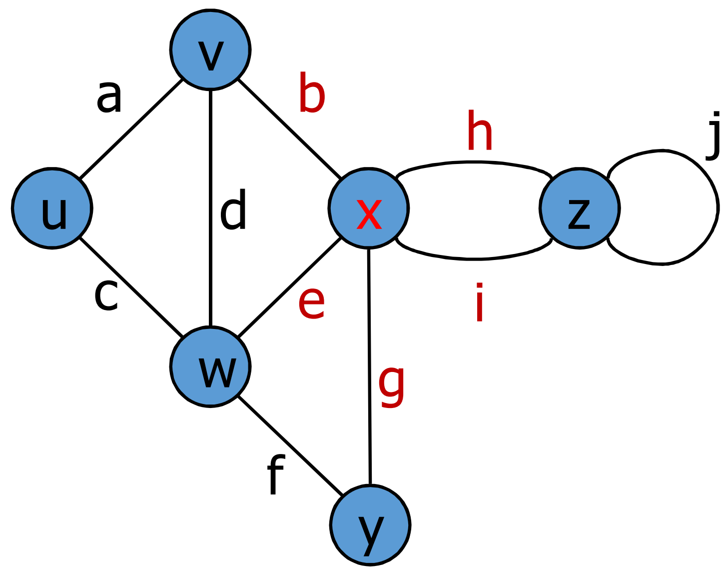
Terminology

- End vertices (or endpoints) of an edge
  - *u* and *v* are the *endpoints* of *a*
- Edges incident to a vertex
  - *a*, *d*, and *b* are *incident* to *v*
- Adjacent vertices
  - u, x and w are *adjacent vertices of v*



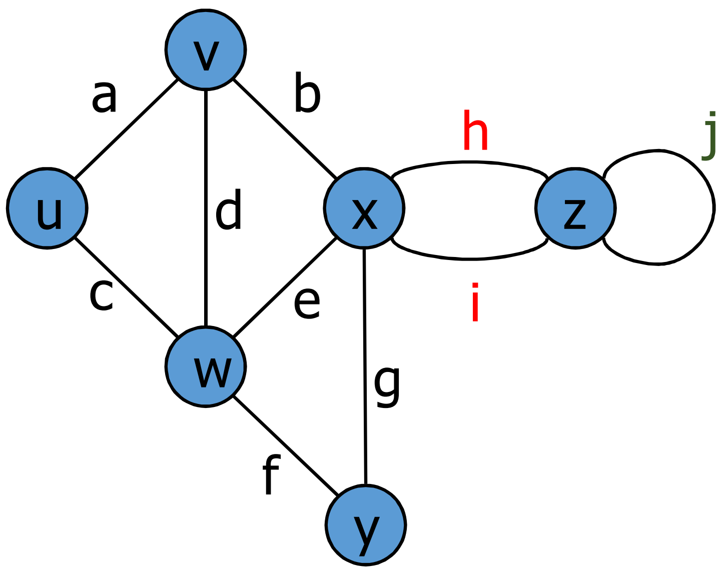
Terminology

- End vertices (or endpoints) of an edge
  - *u* and *v* are the *endpoints* of *a*
- Edges incident to a vertex
  - *a, d*, and *b* are *incident* to *v*
- Adjacent vertices
  - *u* and *v* are *adjacent*
- Degree of a vertex
  - *x* has *degree* 5



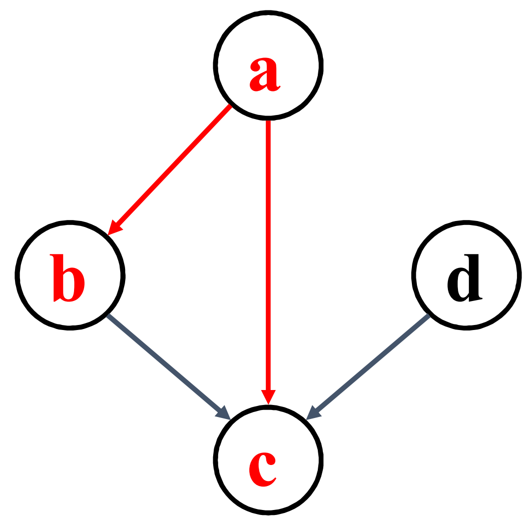
Terminology

- Adjacent vertices
  - *u* and *v* are *adjacent*
- Degree of a vertex
  - *x* has *degree* 5
- Parallel edges
  - *h* and *i* are *parallel edges*
- Self-loop
  - *j* is a *self-loop*



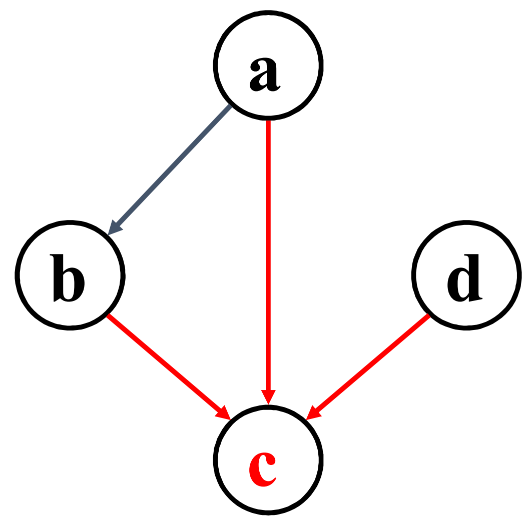
Terminology (cont.)

- Outgoing edges of a vertex
  - (a, b) and (a, c) are outgoing edges of vertex a



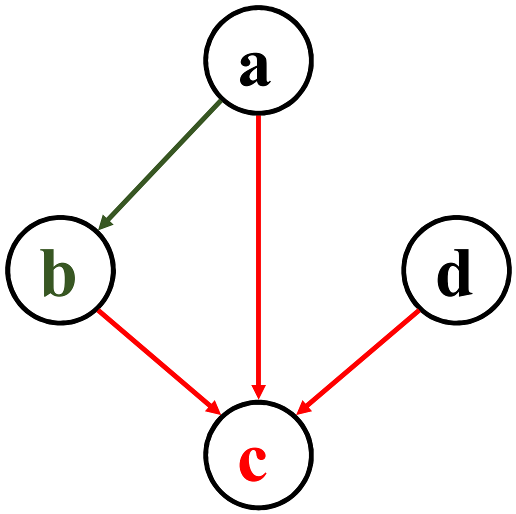
Terminology (cont.)

- Outgoing edges of a vertex
  - (a, b) and (a, c) are outgoing edges of vertex a
- Incoming edges of a vertex
  - (b, c), (d, c) and (a, c) are incoming edges of vertex c



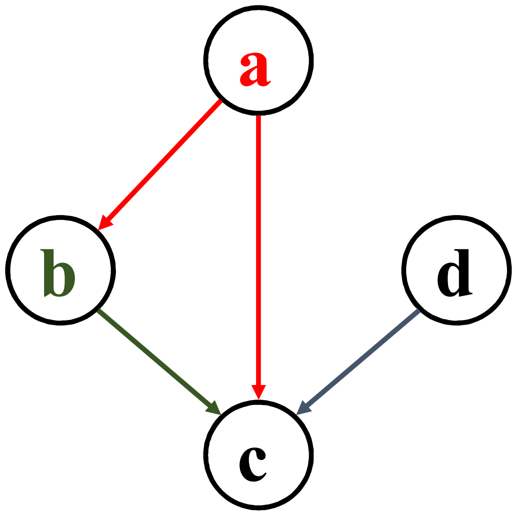
Terminology (cont.)

- Outgoing edges of a vertex
  - (a, b) and (a, c) are outgoing edges of vertex a
- Incoming edges of a vertex
  - (b, c), (d, c) and (a, c) are incoming edges of vertex c
- In-degree of a vertex
  - c has *in-degree* 3
  - b has *in-degree* 1



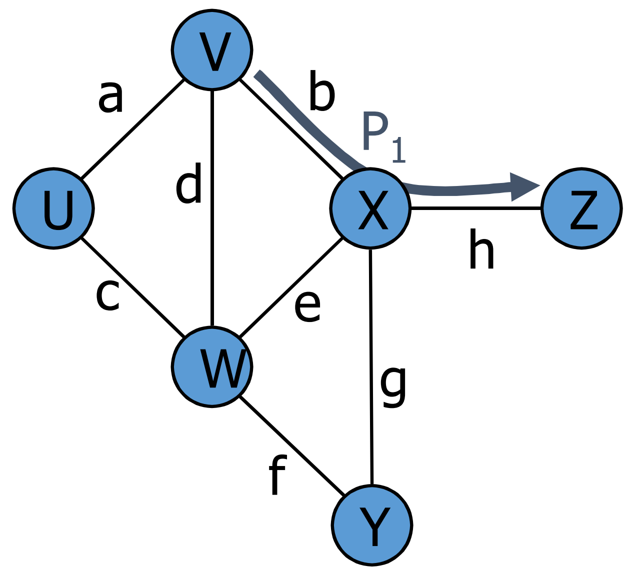
Terminology (cont.)

- Outgoing edges of a vertex
  - (*a*, *b*) and (*a*, *c*) are outgoing edges of vertex *a*
- Incoming edges of a vertex
  - (*b, c*), (*d, c*) and (*a, c*) are incoming edges of vertex *c*
- In-degree of a vertex
  - *c* has *in-degree* 3
  - *b* has *in-degree* 1
- Out-degree of a vertex
  - *a* has *out-degree* 2
  - *b* has *out-degree* 1



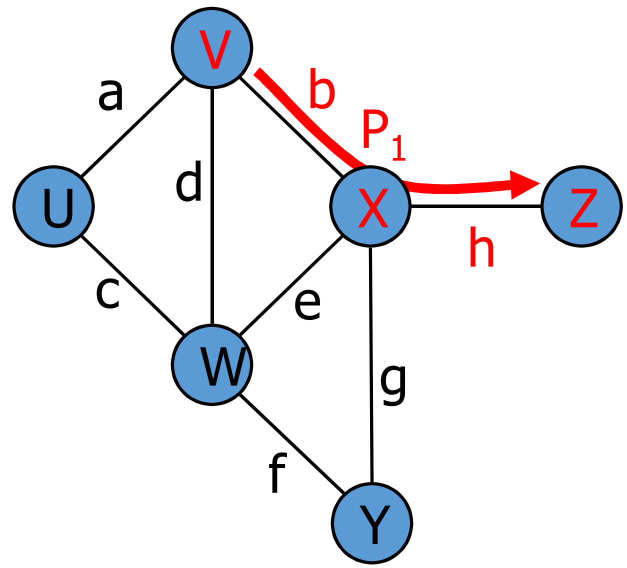
Terminology (cont.)

- Path
  - sequence of alternating vertices and edges
  - begins with a vertex
  - ends with a vertex
  - each edge is preceded and followed by its endpoints



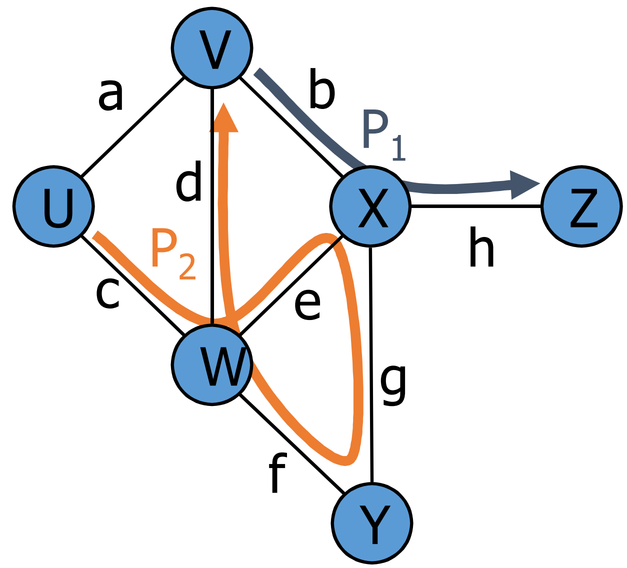
Terminology (cont.)

- Path
  - sequence of alternating vertices and edges
  - begins with a vertex
  - ends with a vertex
  - each edge is preceded and followed by its endpoints
- Simple path
  - path such that all its vertices and edges are distinct
- Examples
  - $P_1=(V, b, X, h, Z)$ is a simple path



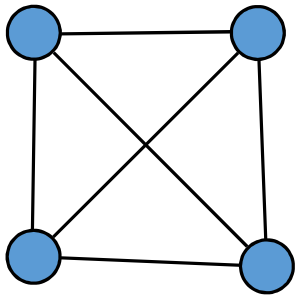
Terminology (cont.)

- Simple path
  - path such that all its vertices and edges are distinct
- Examples
  - $P_1=(V, b, X, h, Z)$ is a simple path
  - $P_2=(U, c, W, e, X, g, Y, f, W, d, V)$ is a path that is not simple



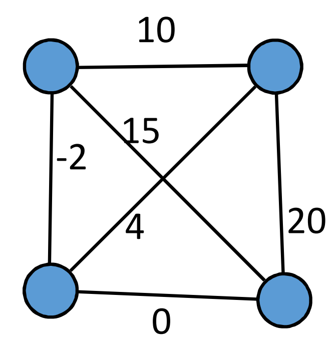
Terminology (cont.)

- *Dense* graph: $|E| \approx |V|^2$; *Sparse* graph: $|E| \approx |V|$ or $|E| << |V|$



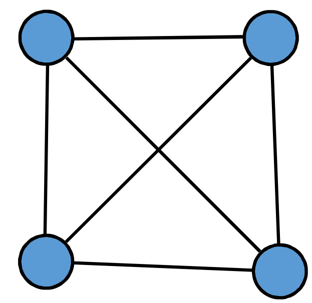
Terminology (cont.)

- *Dense* graph: $|E| \approx |V|^2$; *Sparse* graph: $|E| \approx |V|$ or $|E| << |V|$
- A *weighted graph* associates weights with either the edges or the vertices



Terminology (cont.)

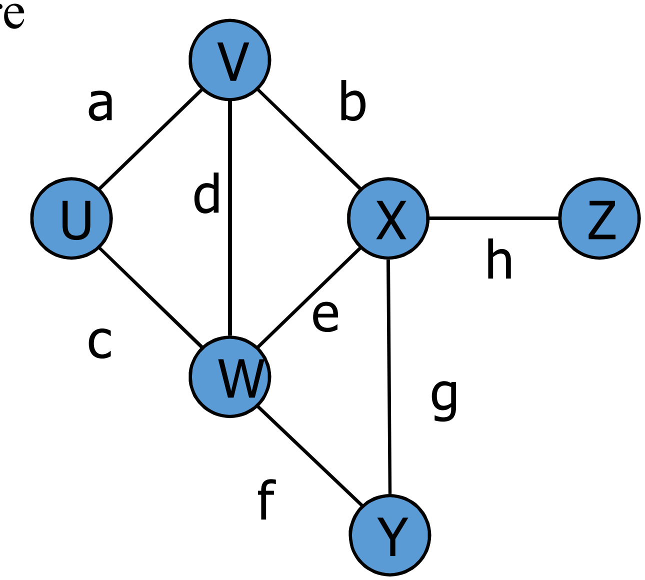
- *Dense* graph: $|E| \approx |V|^2$; *Sparse* graph: $|E| \approx |V|$ or $|E| << |V|$
- A *weighted graph* associates weights with either the edges or the vertices
- A complete graph is a graph that has the maximum number of edges
  - for undirected graph with n vertices, the maximum number of edges is $n(n\text{-}1)/2$
  - for directed graph with n vertices, the maximum number of edges is $n(n\text{-}1)$



A complete undirected graph

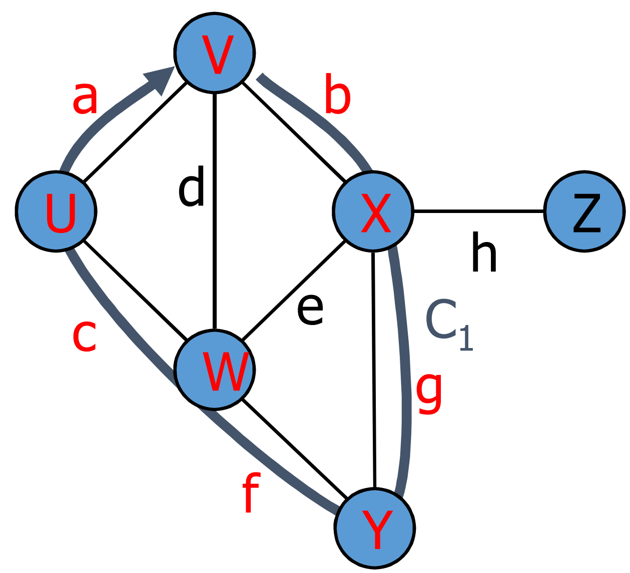
Terminology (cont.)

- Cycle
  - A cycle is a path whose start and end vertices are the same
  - each edge is preceded and followed by its endpoints



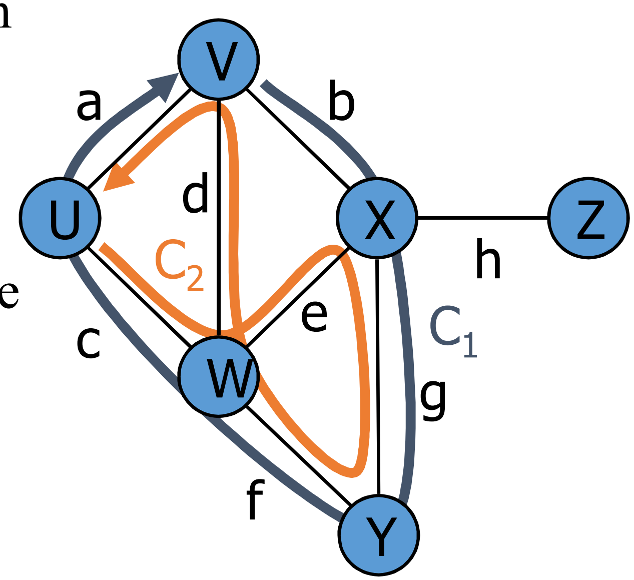
Terminology (cont.)

- Cycle
  - A cycle is a path whose start and end vertices are the same
  - each edge is preceded and followed by its endpoints
- Simple cycle
  - A cycle is simple if each edge is distinct and each vertex is distinct, except for the first and the last one
- Examples
  - $C_1=(V, b, X, g, Y, f, W, c, U, a, V)$ is a simple cycle



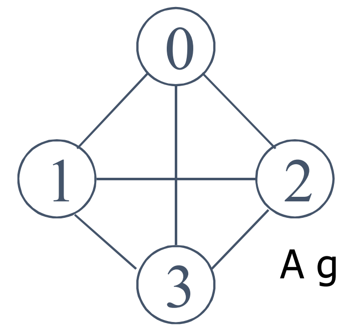
Terminology (cont.)

- Simple cycle
  - A cycle is simple if each edge is distinct and each vertex is distinct, except for the first and the last one
- Examples
  - $C_1=(V, b, X, g, Y, f, W, c, U, a, V)$ is a simple cycle
  - $C_2=(U, c, W, e, X, g, Y, f, W, d, V, a, U)$ is a cycle that is not simple

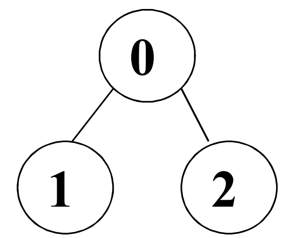


Terminology (cont.)

- A subgraph of G is a graph G' such that
  - $V(G')$ is a subset of $V(G)$ [$V(G') \subseteq V(G)$] and
  - $E(G')$ is a subset of $E(G)$ [$E(G') \subseteq E(G)$]



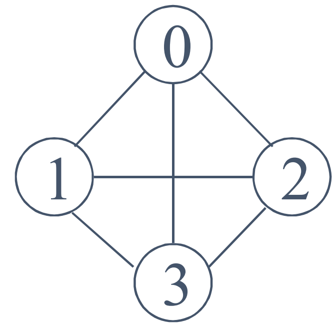
A graph, G

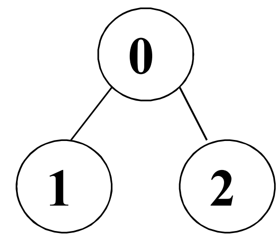


A subgraph, G′

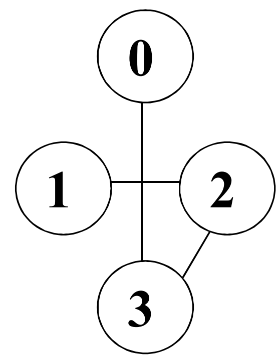
Terminology (cont.)

- A subgraph of G is a graph G’ such that
  - V(G’) is a subset of V(G) [V(G’) $\subseteq$ V(G)] and
  - E(G’) is a subset of E(G) [E(G’) $\subseteq$ E(G)]
- A spanning subgraph G’ of G is a subgraph of G that contains all the vertices of G, that is
  - V(G’) is equal to V(G) [V(G’) = V(G)] and
  - E(G’) is a subset of E(G) [E(G’) $\subseteq$ E(G)]





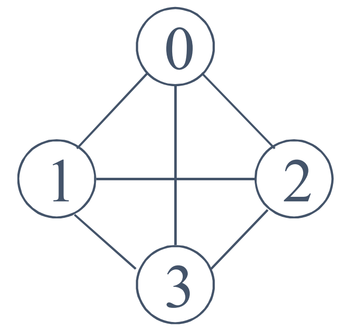
A subgraph

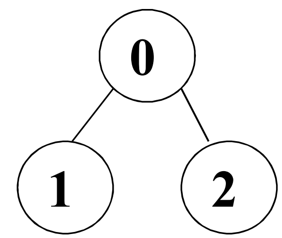


A spanning subgraph (tree)

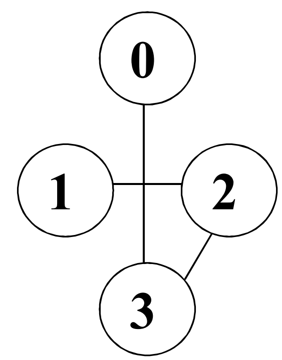
Terminology (cont.)

- A subgraph of G is a graph G' such that
  - V(G') is a subset of V(G) [$V(G') \subseteq V(G)$] and
  - E(G') is a subset of E(G) [$E(G') \subseteq E(G)$]
- A spanning subgraph G' of G is a subgraph of G that contains all the vertices of G, that is
  - V(G') is equal to V(G) [V(G') = V(G)] and
  - E(G') is a subset of E(G) [$E(G') \subseteq E(G)$]
- A forest is a graph without cycles.
- A (free) tree is a connected forest, that is, a connected graph without cycles.
- A spanning tree of a graph G is a spanning subgraph that is a (free) tree.





A subgraph



A spanning subgraph (tree)

Properties

Property 1

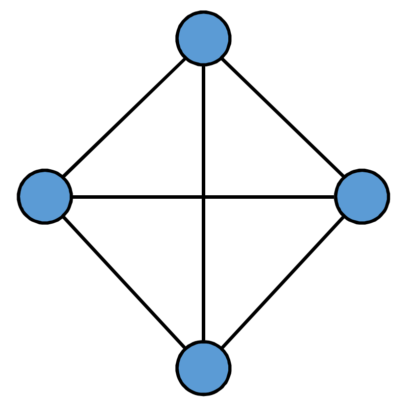
For an undirected graph

$$\Sigma_v \deg(v) = 2m$$

Proof: each edge is counted twice

Notation

n	number of vertices
m	number of edges
$\deg(v)$	degree of vertex v



Properties

Property 1

For an undirected graph

$$\Sigma_v \deg(\boldsymbol{v}) = 2\boldsymbol{m}$$

Proof: each edge is counted twice

Property 2

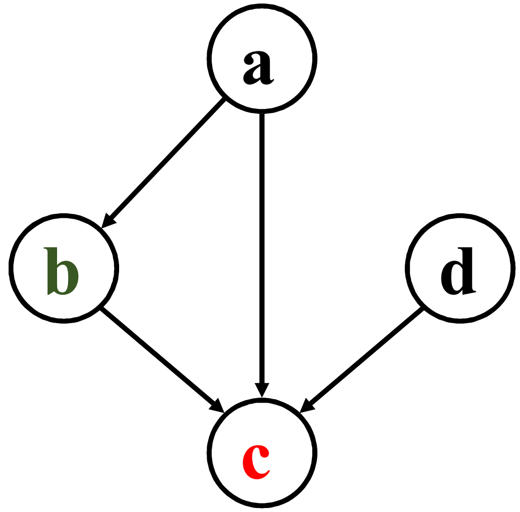
For a directed graph

$$\Sigma_v \operatorname{indeg}(\boldsymbol{v}) = \Sigma_v \operatorname{outdeg}(\boldsymbol{v}) = \boldsymbol{m}$$

Proof: each edge is counted once for in-degree and once for out-degree

Notation

$\boldsymbol{n}$	number of vertices
$\boldsymbol{m}$	number of edges
$\deg(\boldsymbol{v})$	degree of vertex $\boldsymbol{v}$



Properties

Property 2

For a directed graph

$$\Sigma_v \, \mathrm{indeg}(\boldsymbol{v}) = \Sigma_v \, \mathrm{outdeg}(\boldsymbol{v}) = \boldsymbol{m}$$

Proof: each edge is counted once for in-degree and once for out-degree

Notation

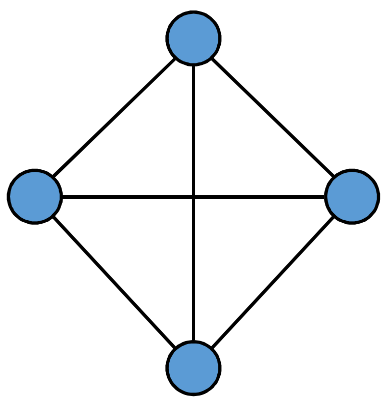
$\boldsymbol{n}$	number of vertices
$\boldsymbol{m}$	number of edges
$\deg(\boldsymbol{v})$	degree of vertex $\boldsymbol{v}$

Property 3

If G is a simple undirected graph, then $\boldsymbol{m} \leq \boldsymbol{n}\,(\boldsymbol{n}-1)/2$,and

if G is a simple directed graph, then $\boldsymbol{m} \leq \boldsymbol{n}\,(\boldsymbol{n}-1)$.

Proof: each vertex has degree at most $(\boldsymbol{n}-1)$. Then use Property 1 and Property 2.

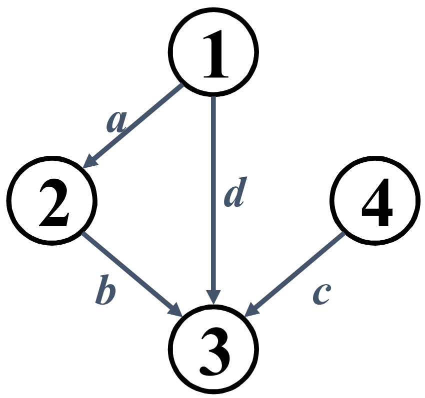


Graph Representation

- For graphs to be computationally useful, they have to be conveniently represented in programs
- There are two computer representations of graphs:
  - **Adjacency matrix representation**
  - **Adjacency lists representation**

Adjacency Matrix Representation

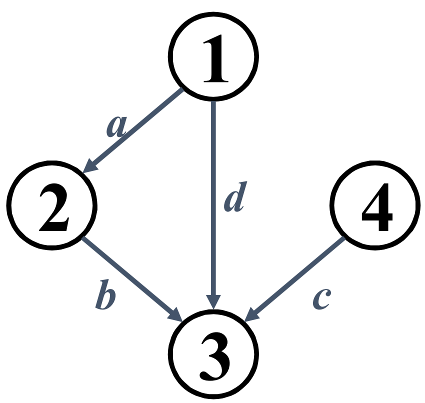
- Assume V = {1, 2, …, n}
- An *adjacency matrix* represents the graph as a n x n matrix A:
  - A[i, j] = 1 if edge $(i, j) \in E$ (or weight of edge)
    = 0 if edge $(i, j) \notin E$



A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

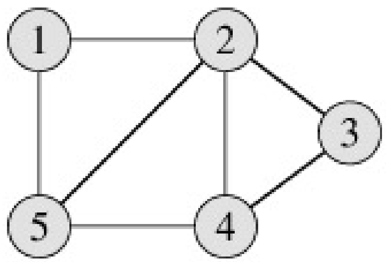
Adjacency Matrix Representation

- Assume V = {1, 2, …, n}
- An *adjacency matrix* represents the graph as a n x n matrix A:
  - A[i, j] = 1 if edge (i, j) $\in$ E (or weight of edge)
    = 0 if edge (i, j) $\notin$ E



A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

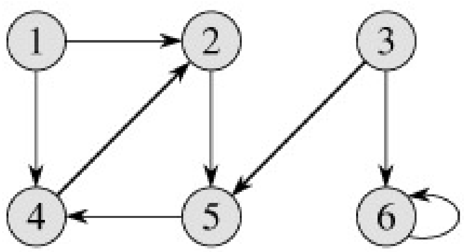
Adjacency Matrix Representation



Undirected Graph

$$\begin{array}{c|ccccc|} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 & 1 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 \\ \hline \end{array}$$

Symmetric matrix



Directed Graph

$$\begin{array}{c|cccccc|} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 1 & 0 \\ 3 & 0 & 0 & 0 & 0 & 1 & 1 \\ 4 & 0 & 1 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 1 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline \end{array}$$

may NOT be symmetric

Adjacency Matrix Representation

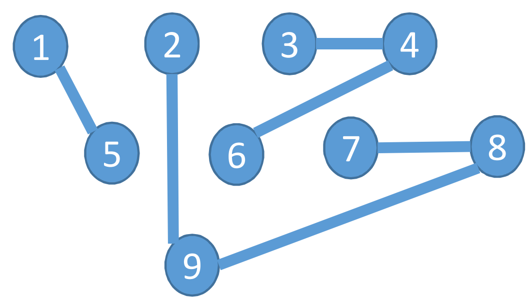
- Pros:
  - Simple to implement
  - Easy and fast to tell if a pair (i, j) is an edge: simply check if A[i, j] is 1 or 0
  - Can be very efficient for small graphs
  - Good for dense graphs (why?)
- Cons:
  - No matter how few edges the graph has, the matrix takes $O(n^2)$, i.e., $O(|V|^2)$ in memory

|V| = n

|V| = n

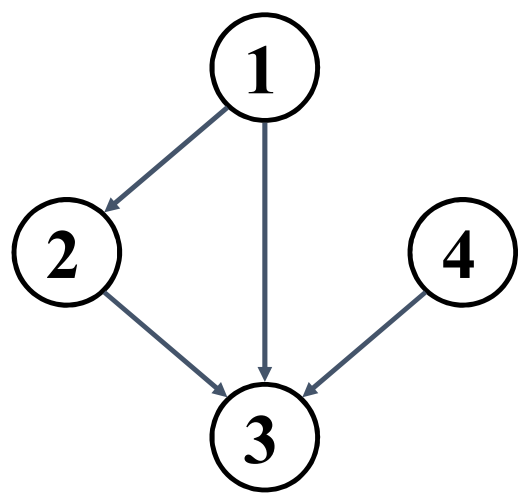
	1	2	3	4	5	6	7	8	9
1	0	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	0	0	1
3	0	0	0	1	0	0	0	0	0
4	0	0	1	0	0	1	0	0	0
5	1	0	0	0	0	0	0	0	0
6	0	0	0	1	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	0	1	0	1
9	0	1	0	0	0	0	0	1	0

$n \times n$ matrix

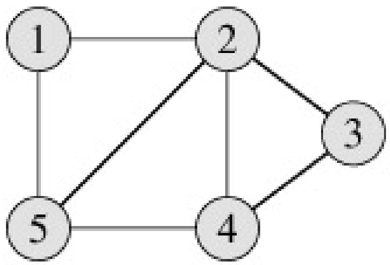


Adjacency Lists Representation

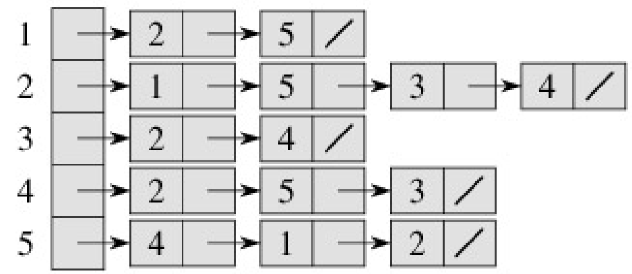
- A graph is represented by a one-dimensional array L of linked lists, where
  - L[i] is the linked list containing all the nodes adjacent to node i.
  - The nodes in the list L[i] are in NO particular order

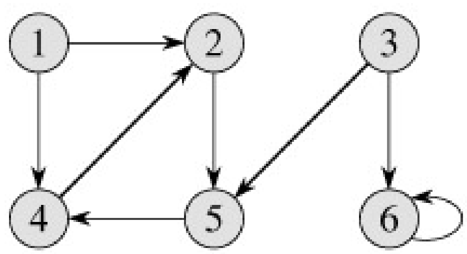


Adjacency Lists Representation

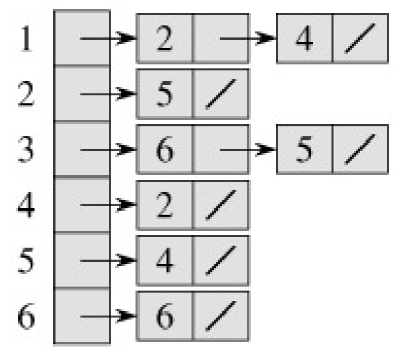


Undirected Graph





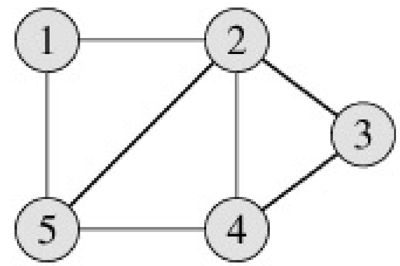
Directed Graph



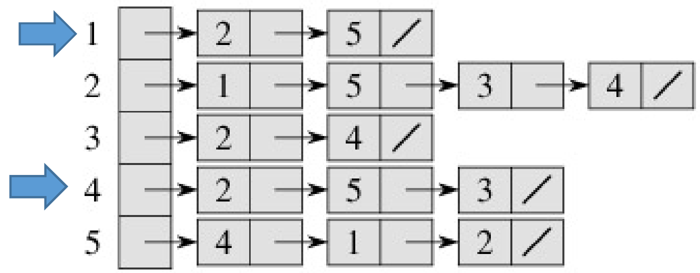
Adjacency Lists Representation

- Pros:
  - Saves on space (memory): the representation takes O(|V|+|E|) memory.
  - Good for large, sparse graphs (e.g., planar maps)

How to find whether there is an edge (4,1)?

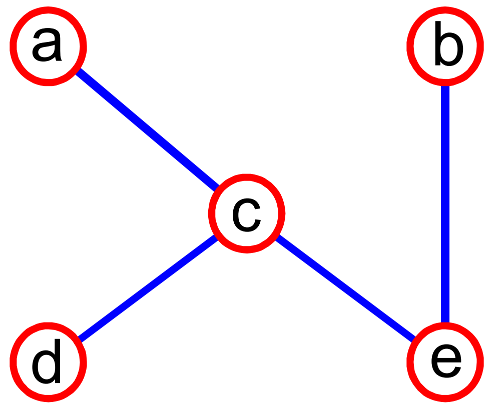


- Cons:
  - It can take up to O(n) time to determine if a pair of nodes (i, j) is an edge: one would have to search the linked list L[i], which takes time proportional to the length of L[i].



Tree: Definition

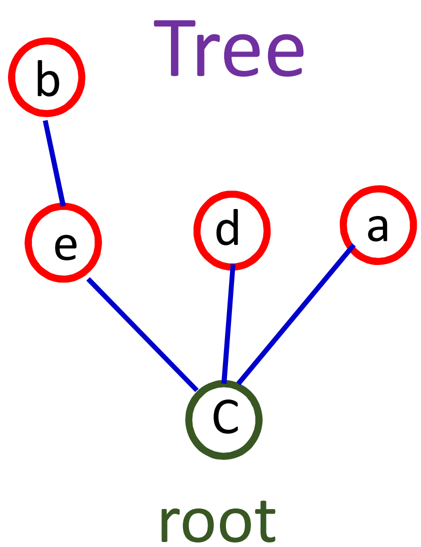
- A tree is a special type of graph!
- A tree is a graph that is connected and acyclic.
- A tree consists of one or more nodes
- a free tree has NO special node.

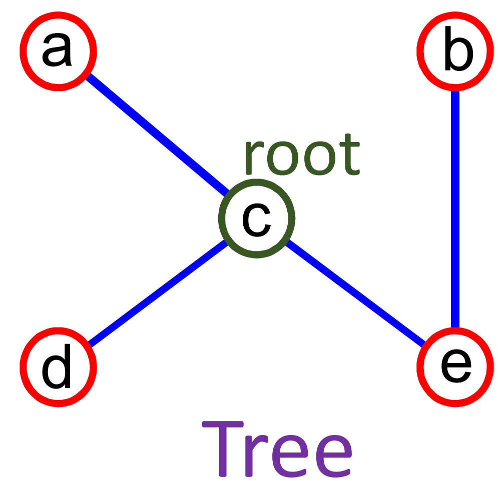


Tree

Rooted Tree: Definition

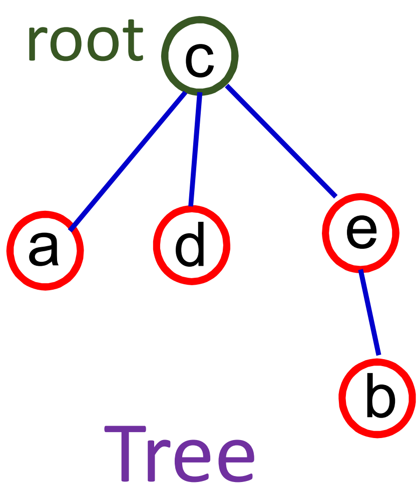
- a rooted tree has a special node, e.g., first node or root.
- every node has a parent except the root
- every node has zero or more children

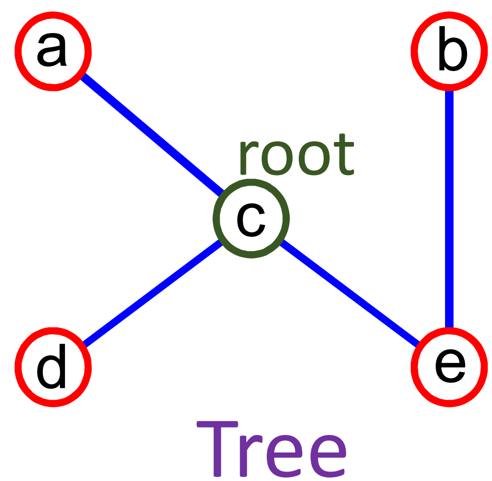




Rooted Tree: Definition

- a rooted tree has a special node, e.g., first node or root.
- every node has a parent except the root
- every node has zero or more children





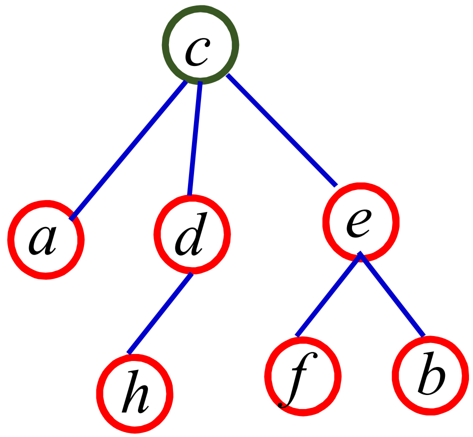
Rooted Tree: Definition

degree of a node:

No. of children of a node

degree (c) =3, degree (e) =2 , degree (d) =1

others have degree 0



Rooted Tree: Definition

- **Leaf** nodes: nodes with degree 0: *a, d, f, b*
- **Internal** nodes: other nodes : *c, e*