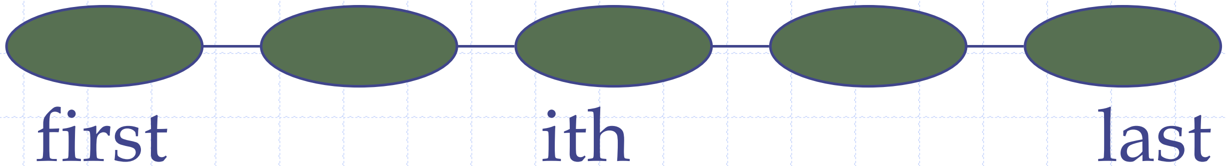
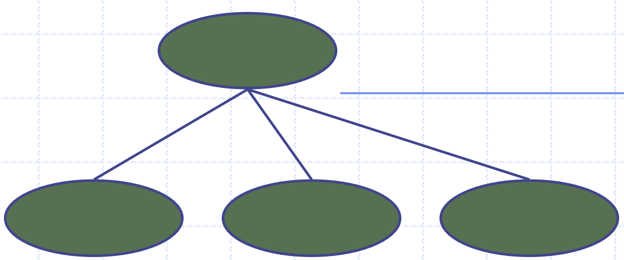
Graphs

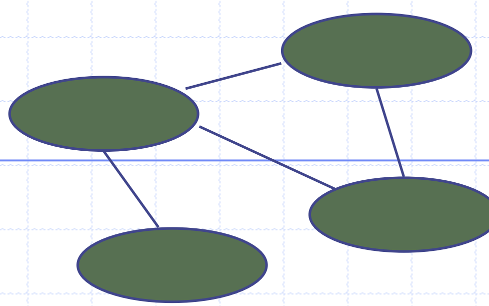
sequence/linear (1 to 1)



hierarchical
(1 to many)

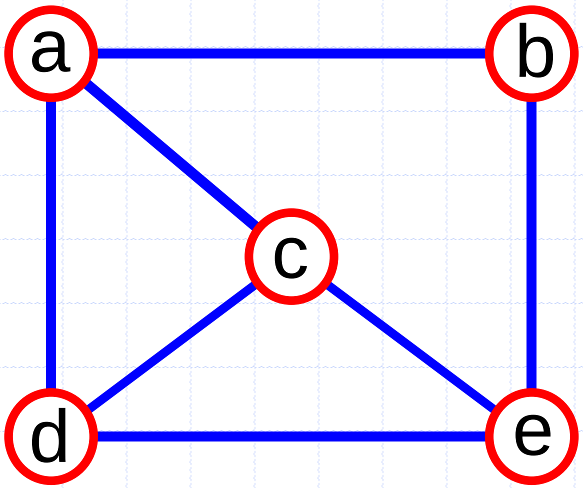


graph (many to many)



What is a Graph?

- 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:



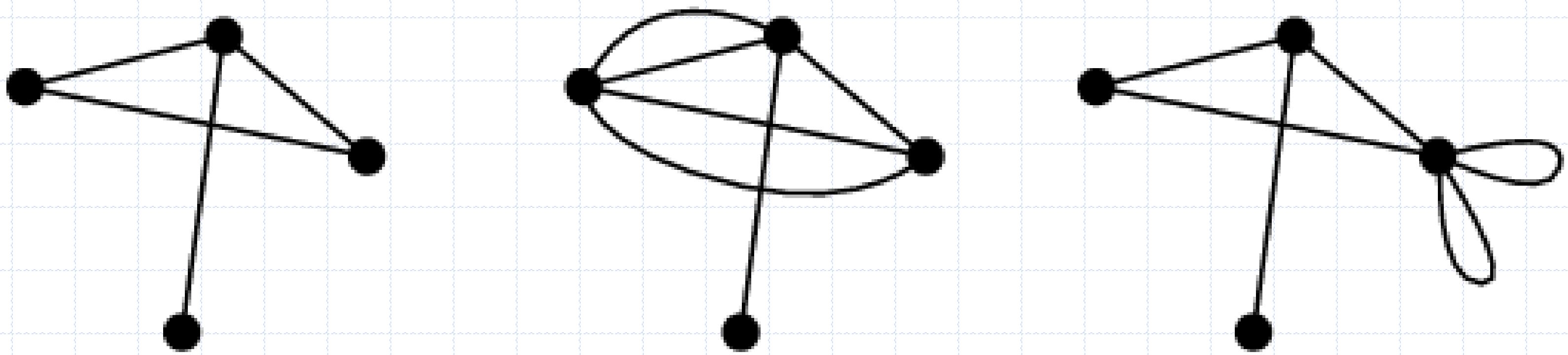
V= {a, b, c, d, e}

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

- **A tree is a special type of graph!**

What is a Graph?

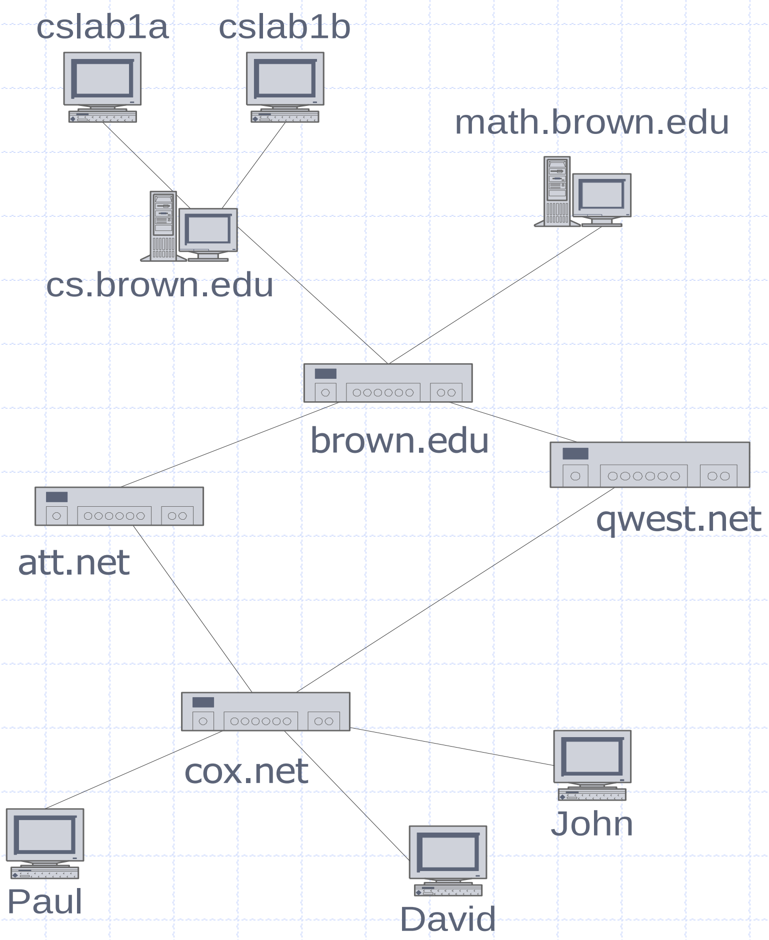
- A **simple graph**, also called a strict **graph** is an unweighted, undirected **graph** containing no **graph** loops or multiple edges



simple graph — *nonsimple graph with multiple edges* — *nonsimple graph with loops*

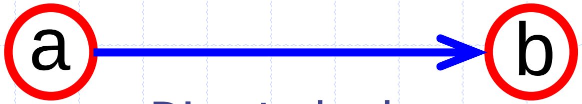
Applications

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

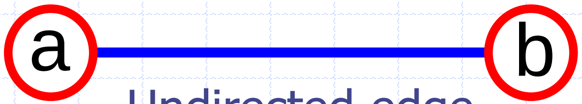


Edge and Graph Types

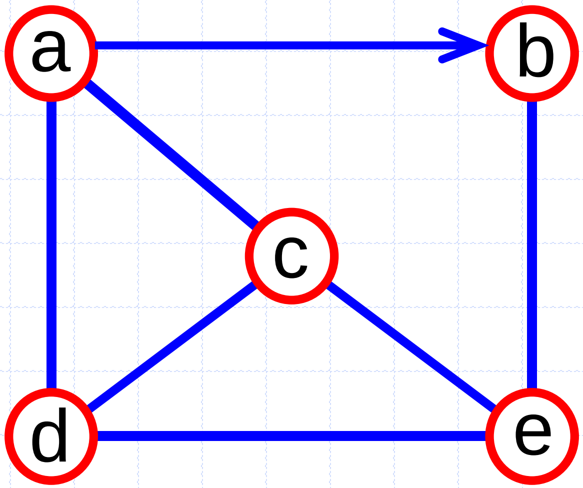
- Directed edge
  - ordered pair of vertices (u,v)
  - first vertex u is the origin
  - second vertex v is the destination
- Undirected edge
  - unordered pair of vertices (u,v)
- Directed graph (Digraph)
  - all the edges are directed
  - e.g., route network
- Undirected graph
  - all the edges are undirected
  - e.g., flight network
- Mixed graph
  - some edges are undirected and some edges are directed
  - e.g., a graph modeling a city map



Directed edge



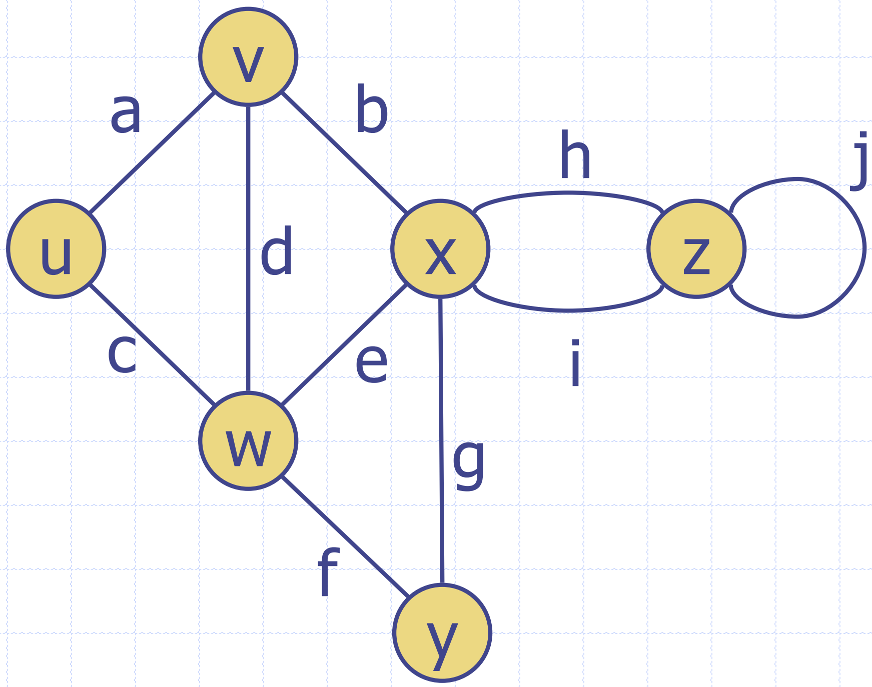
Undirected edge



Mixed graph

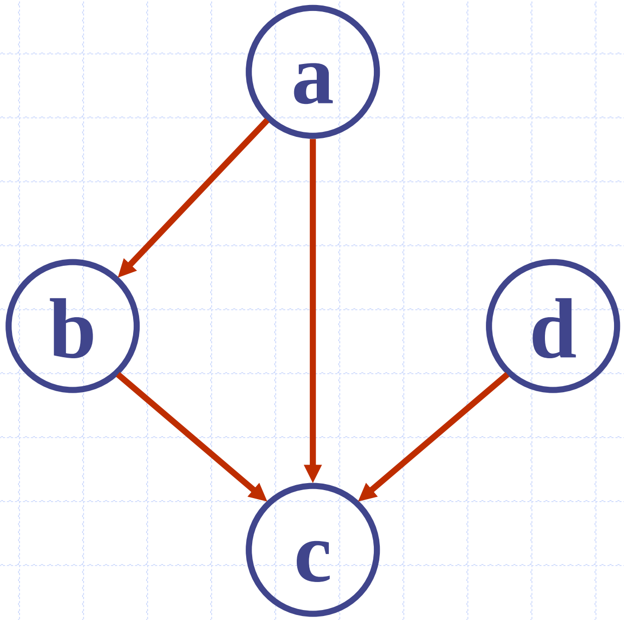
Terminology

- 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
- Parallel edges
  - h and i are *parallel edges*
- Self-loop
  - j is a *self-loop*



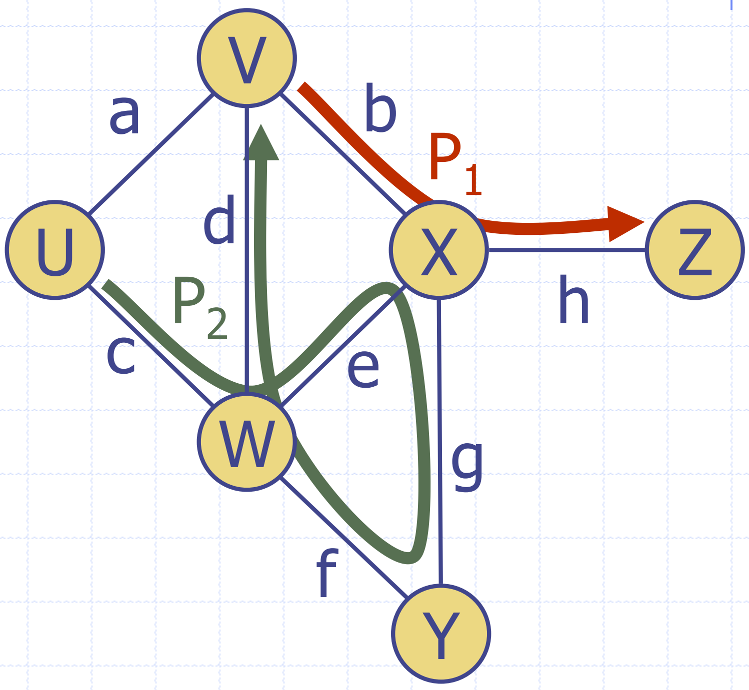
Terminology (cont.)

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



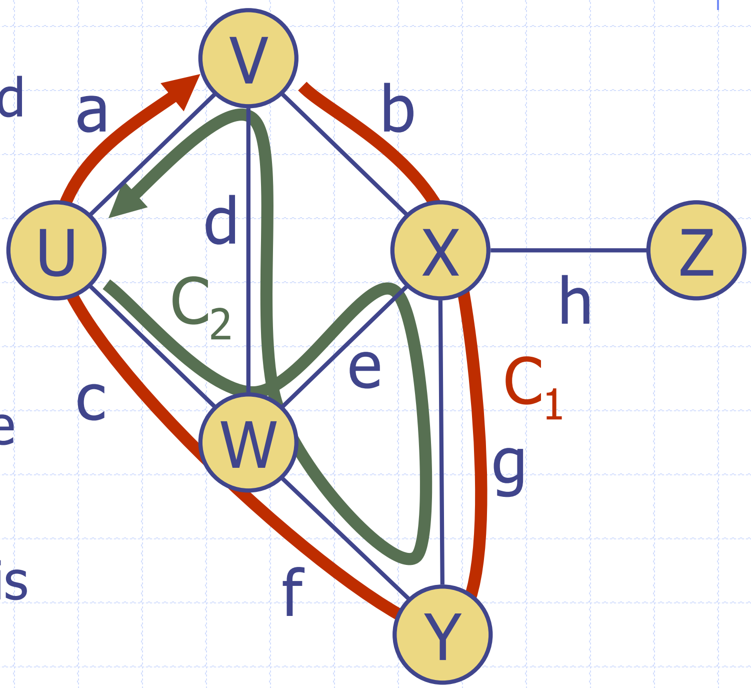
Terminology (cont.)

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



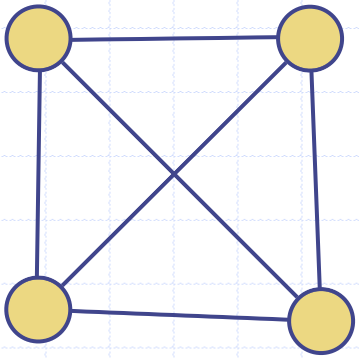
Terminology (cont.)

- 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
  - C_2=(U,c,W,e,X,g,Y,f,W,d,V,a,U) is a cycle that is not simple



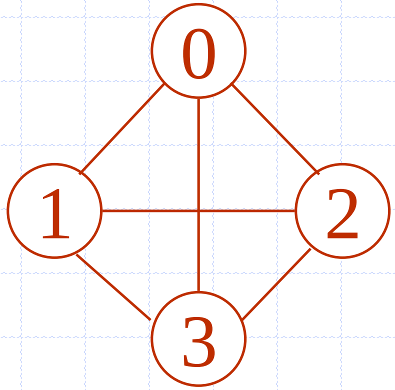
Terminology (cont.)

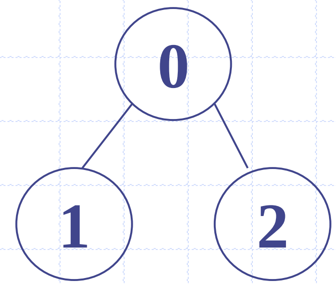
- *Dense* graph: $|E| \approx |V|^2$; *Sparse* graph: $|E| \approx |V|$
- A *weighted graph* associates weights with either the edges or the vertices
- A complete graph is a graph that has the maximum number of edges
  - for undirected graph with n vertices, the maximum number of edges is $n(n-1)/2$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



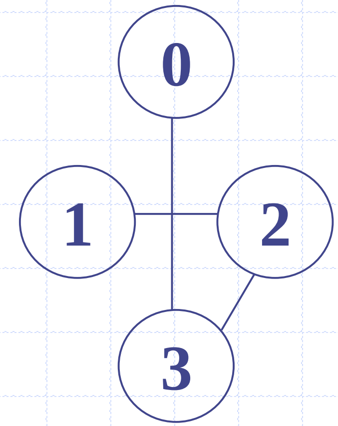
Terminology (cont.)

- 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.



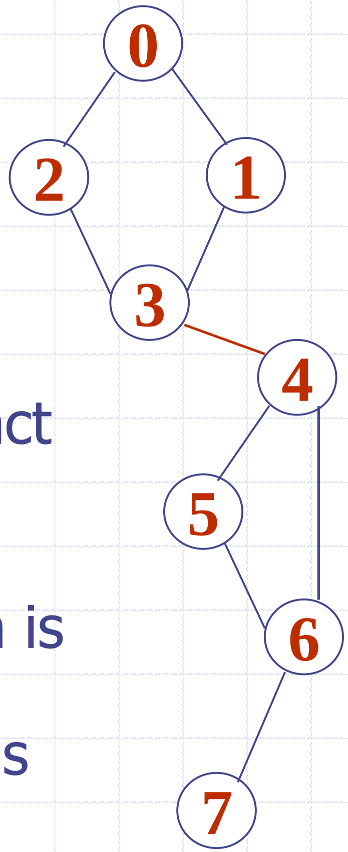


A subgraph

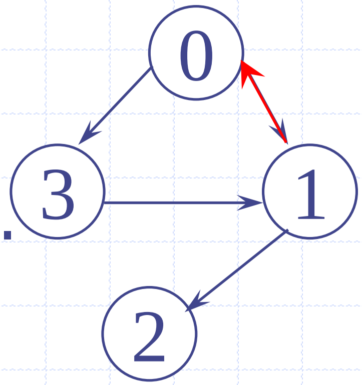


A spanning subgraph (tree)

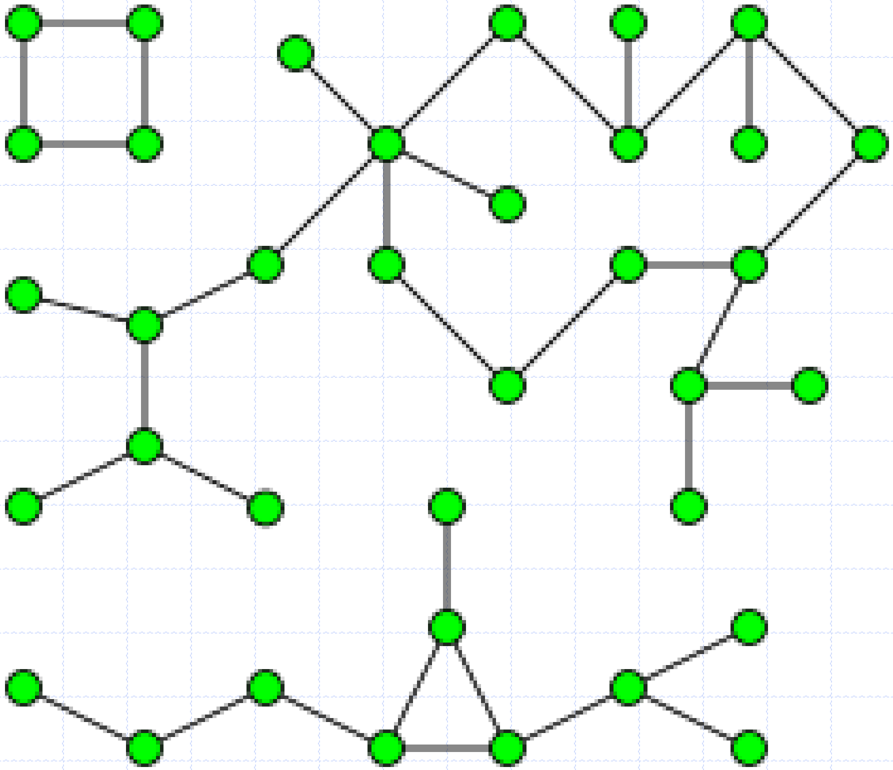
Terminology (cont.)



- In a graph G, two vertices, v_0 and v_1, are connected if there is a path in G from v_0 to v_1
- A graph is connected if, for every pair of distinct vertices v_i and v_j, there is a path from v_i to v_j
- A component (sometimes referred to as connected component) of an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph.
- A tree is a graph that is connected and acyclic.
- A directed graph is strongly connected if there is a directed path from v_i to v_j and also from v_j to v_i.
- A strongly connected component is a maximal subgraph that is strongly connected.



Terminology (cont.)



A graph with three components.

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

Properties

Property 1

For an undirected graph

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

Proof: each edge is counted twice

Property 2

For a directed graph

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

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

Notation

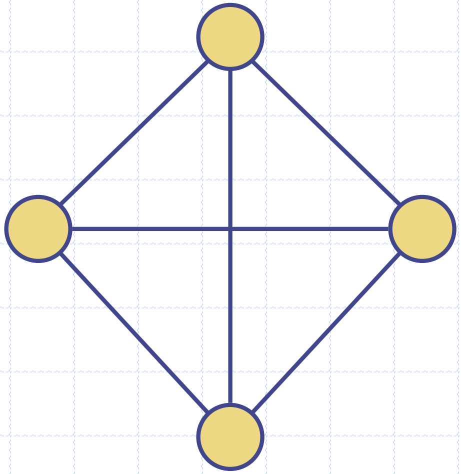
n	number of vertices
m	number of edges
$\deg(v)$	degree of vertex v

Property 3

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

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

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

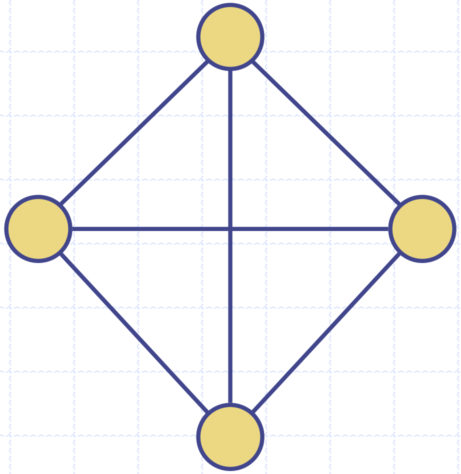


Properties

Property 4

Let G be an undirected graph with n vertices and m edges. Then we have the following:

- If G is connected, then $m \geq n - 1$,
- If G is a tree, then $m = n - 1$,
- If G is a forest, then $m \leq n - 1$.



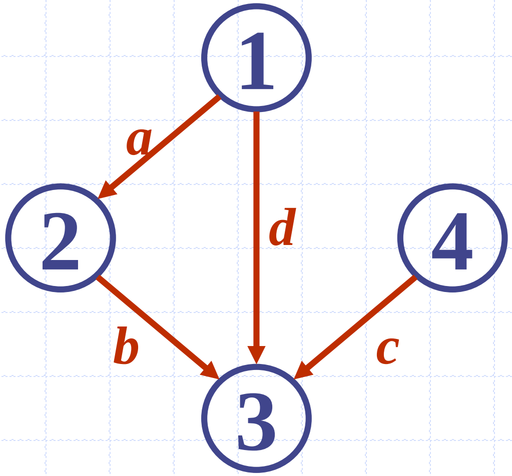
Proof: Let G be a connected graph. Delete edges one by one from G keeping G connected. At last there will remain $n - 1$ edges.

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

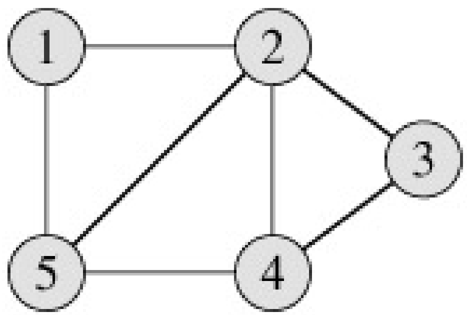
Adjacency Matrix Representation

- 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



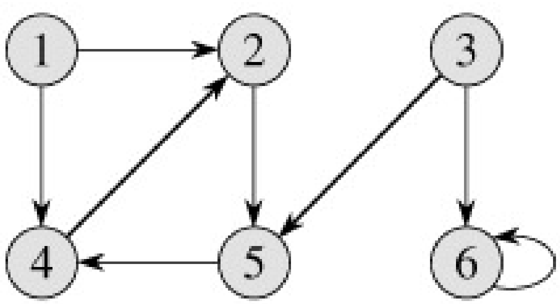
A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

Adjacency Matrix Representation



Undirected Graph

	1	2	3	4	5
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



Directed Graph

	1	2	3	4	5	6
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

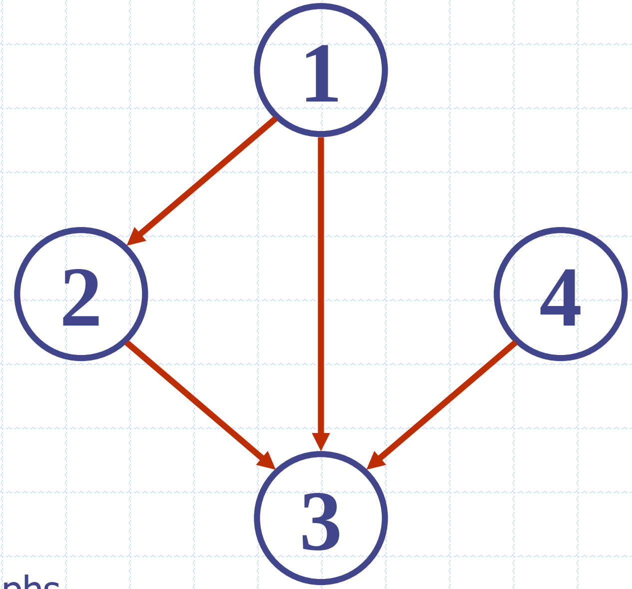
The adjacency matrix for an undirected graph is symmetric; the adjacency matrix for a digraph need not be symmetric

Adjacency Matrix Representation

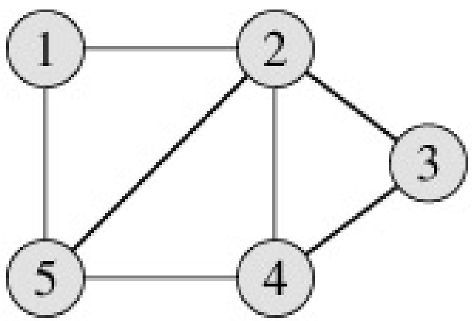
- Pros:
  - Simple to implement
  - Easy and fast to tell if a pair (i, j) is an edge: simply check if A[i, j] is 1 or 0
  - Can be very efficient for small graphs
- Cons:
  - No matter how few edges the graph has, the matrix takes $O(n^2)$ in memory

Adjacency Lists Representation

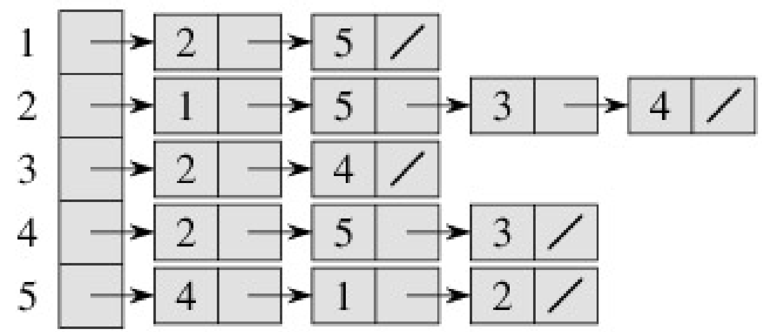
- A graph is represented by a one-dimensional array L of linked lists, where
  - L[i] is the linked list containing all the nodes adjacent to node i.
  - The nodes in the list L[i] are in no particular order
- Example:
  - Adj[1] = {2,3}
  - Adj[2] = {3}
  - Adj[3] = {}
  - Adj[4] = {3}

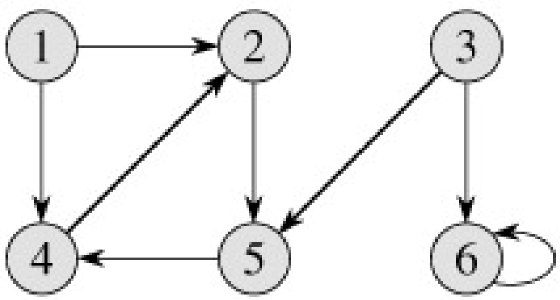


Adjacency Lists Representation

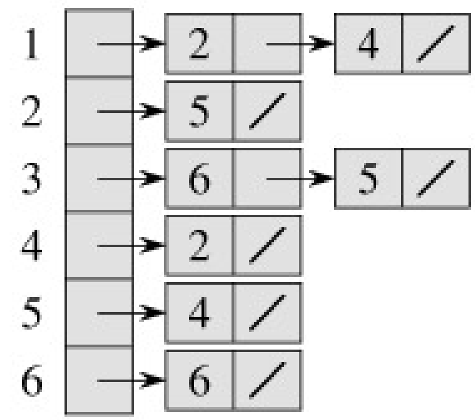


Undirected Graph





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)
- 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 $L[i]$.

Graph ADT members?

Graph ADT members?

- Variables
  - int V;
  - LinkedList * E;
  - bool directed;
- Common Methods:
  - AddEdge(u, v);
  - RemoveEdge(u, v);
  - isEdge(u, v);
  - getAdjacentNodes(u);