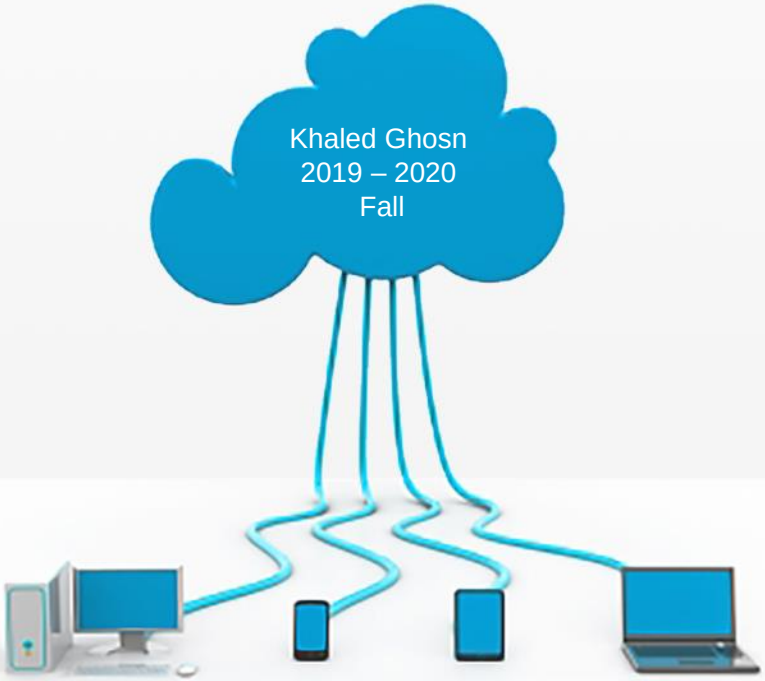
ARTS, SCIENCES & TECHNOLOGY UNIVERSITY IN LEBANON

AUL

Graph

Graph and paths

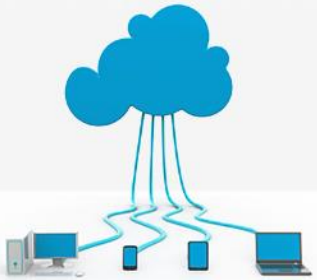


Khaled Ghosn
2019 – 2020
Fall

Graph

Definitions

- ✓ A graph is a collection of vertices (nodes) and edges (arcs)
  - Each vertex contains an element
  - Each edge connects two vertices together
    (or possibly the same vertex to itself)
    and may contain an edge attribute
- ✓ A graph is a way of representing connections or relationships between pairs of objects

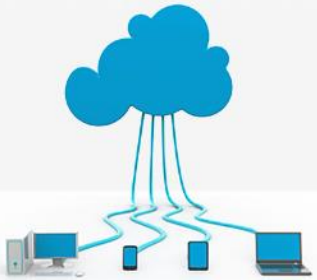


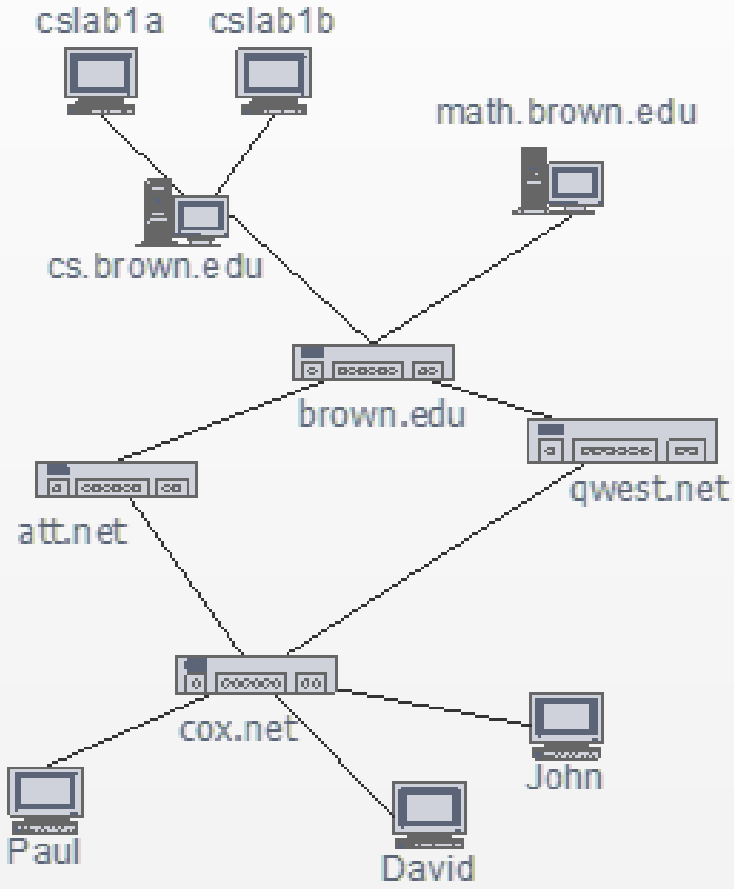
Graph

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



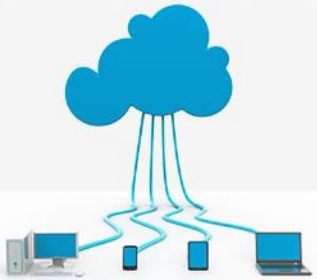


Graph

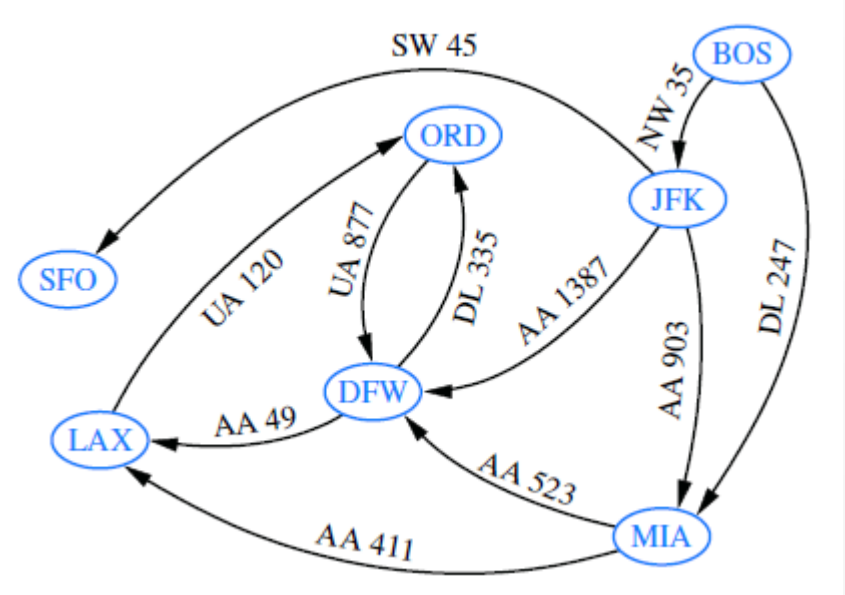
Definitions

- ✓ There are two kinds of graphs:
  - ➢ A Directed graph is one in which the edges have a direction
    - ➢ sometimes called digraphs
    - ➢ Asymmetric relation
    - ➢ ordered pair of nodes
  - ➢ An Undirected graph is one in which the edges do not have a direction
    - ➢ Symmetric relation
    - ➢ unordered pair of nodes
  - ➢ A Mixed graph has both directed and undirected edges

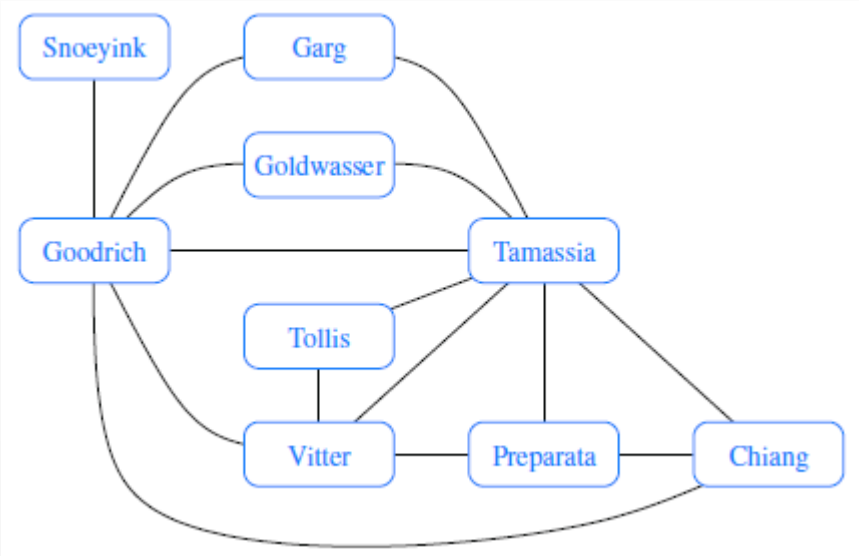
+ Can undirected or mixed graph be converted into a directed graph?



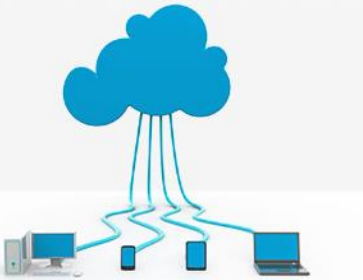
Graph



A directed graph



An undirected graph



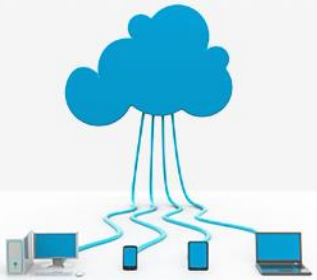
Graph

Terminologies

- ✓ Size of a graph is the number of *vertices* in it
- ✓ Empty graph has size zero (no vertices)

- ✓ Endpoints (end vertices): two vertices joined by an edge
- ✓ Origin: the first endpoint in a directed graph
- ✓ Destination: the second endpoint in a directed graph

- ✓ Adjacent: If two vertices are connected by an edge, they are adjacent to each other (and they are neighbors)
- ✓ Incident edge to a vertex: An edge is said to be incident to a vertex if the vertex is one of the edge's endpoints
- ✓ Outgoing edges of a vertex are the directed edges whose origin is that vertex.
- ✓ Incoming edges of a vertex are the directed edges whose destination is that vertex

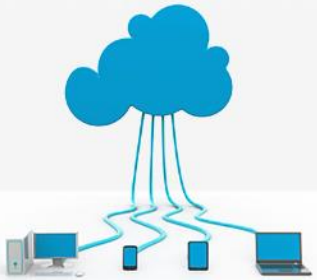


Graph

Terminologies

- ✓ The degree of a vertex is the number of edges it has (is the number of incident edges of the vertex)

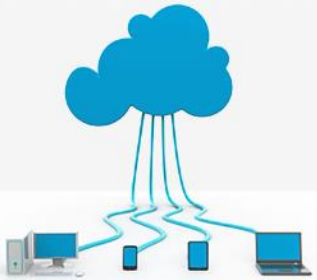
- ✓ For directed graphs:
  - The in-degree of a vertex is the number of in-edges it has
  - The out-degree of a vertex is the number of out-edges it has
  - If a directed edge goes from vertex S to vertex D, we call S the source and D the destination of the edge
    - The edge is an out-edge of S and an in-edge of D
    - S is a predecessor of D, and D is a successor of S



Graph

Terminologies

- ✓ Parallel edges (multiple edges): two directed edges having same origin and same destination
- ✓ Self-Loop edge: is an edge having one vertex as source and destination
- ✓ Simple graph: graph not having parallel edges or self-loops
- ✓ Path: is a list of edges such that each vertex (but not the last) is the predecessor of the next vertex in the list
- ✓ Simple Path: when each vertex in the path is distinct
- ✓ Directed path: is a path such that all edges are directed and are traversed along their direction

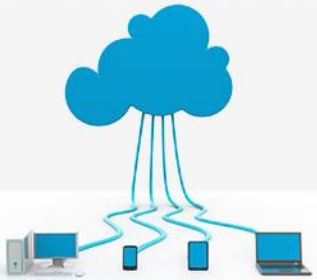


Graph

Terminologies

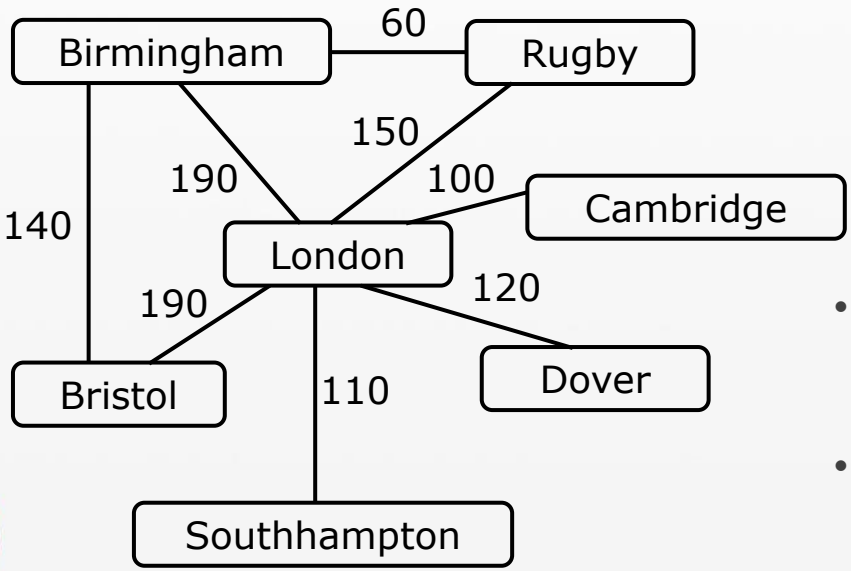
- ✓ Cycle: is a path whose first and last vertices are the same
- ✓ Simple cycle: when each vertex in the cycle is distinct
- ✓ Directed cycle: is a cycle such that all edges are directed and are traversed along their direction

- ✓ Cyclic graph: A graph contains at least one cycle
- ✓ Acyclic graph: A graph does not contain any cycles
  - A directed graph is acyclic if it has no directed cycles

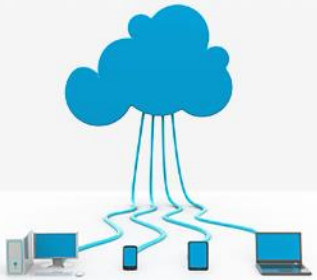


Graph

Terminologies



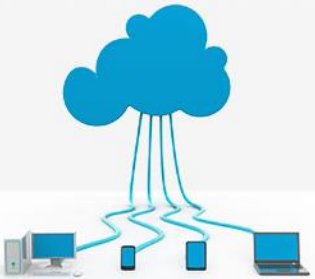
- Example: (Bristol, Birmingham, London, Dover) is a path
- Example: (London, Bristol, Birmingham, London) is a cycle



Graph

Terminologies

- ✓ A graph is connected if there is a path from every vertex to every other vertex (for any two vertices, there is a path between them)
- ✓ A *directed graph* is strongly connected if there is a path from every vertex to every other vertex
- ✓ A directed graph is weakly connected if the underlying undirected graph is connected
- ✓ Vertex X is reachable from vertex Y if there is a path from Y to X



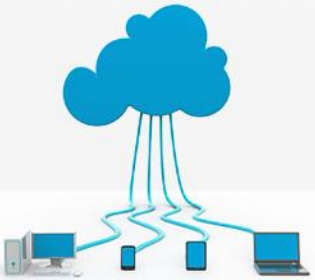
Graph

Terminologies

- ✓ Subgraph (H) of a graph (G): is a graph (H) whose vertices and edges are subsets of the nodes and edges of a graph (G)
- ✓ Spanning subgraph of G is a subgraph of G that contains all the vertices of the graph *G*

- ✓ If a graph (G) is not connected, its maximal connected subgraphs are called the connected components of graph (G)

- ✓ Forest is a graph without cycles
- ✓ Tree is a connected forest, that is, a connected graph without cycles
- ✓ Spanning tree of a graph is a spanning subgraph that is a tree

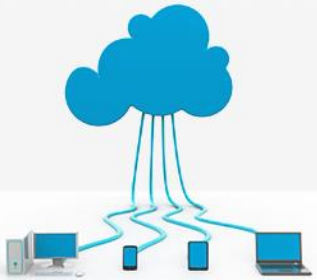


Graph

Data Structures

Four data structures for representing a graph:

1. **Edge List**
   to maintain an unordered list of all edges
2. **Adjacency List**
   to maintain, for each vertex, a separate list containing those edges that are incident to the vertex (allows to find - more efficiently - find all edges incident to a given node)
3. **Adjacency map**
   is similar to an adjacency list, but the secondary container of all edges incident to a vertex is organized as a map, rather than as a list, with the adjacent vertex serving as a key. This allows more efficient access to a specific edge
4. **Adjacency matrix**
   provides worst-case O(1) access to a specific edge by maintaining an n×n matrix, for a graph with n vertices

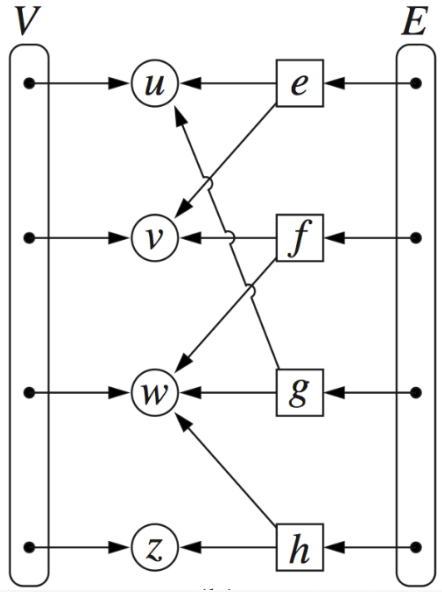


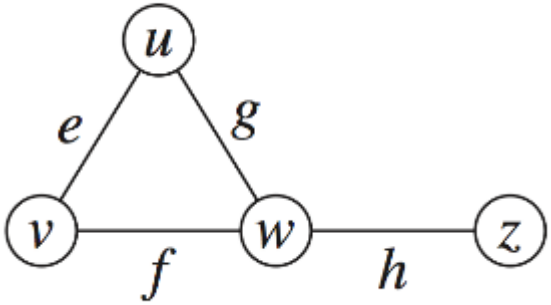
Edge List Structure

Data Structures

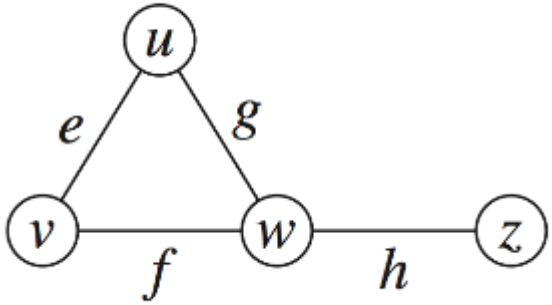
- ✓ The simplest
- ✓ Not the most efficient

- ✓ Vertices are stored in an unordered list (DL
- ✓ Edges are stored in an unordered list (DLL)





Edge List Structure



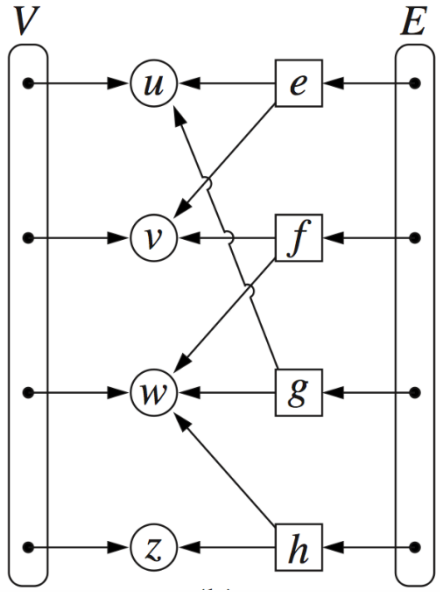
Data Structures

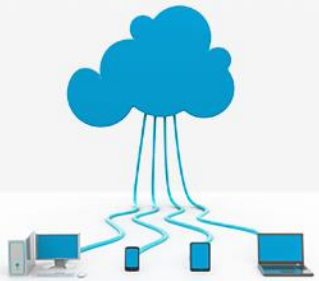
Vertex object
- element
- reference to position in vertex sequence

Edge object
- element
- origin vertex object
- destination vertex object
- reference to position in edge sequence

Vertex sequence
- sequence of vertex objects

Edge sequence
- sequence of edge objects



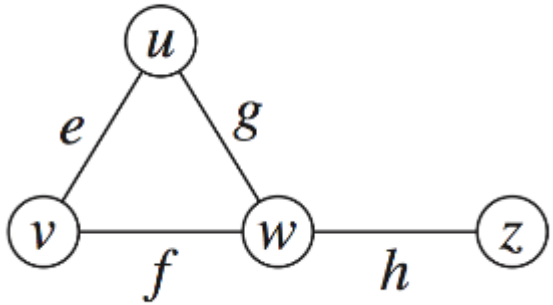


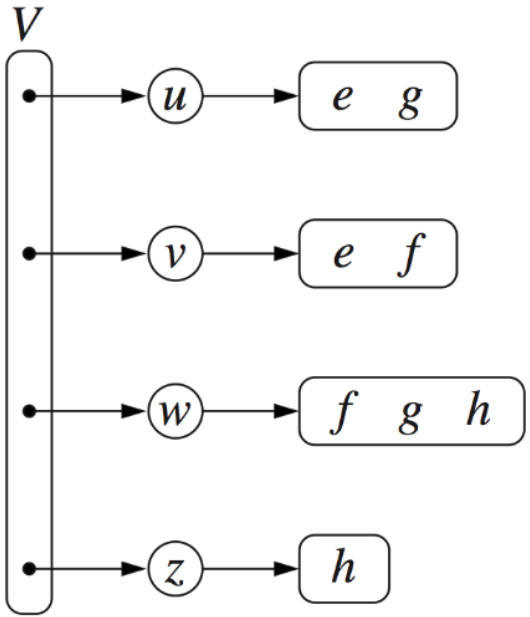
Adjacency List Structure

Data Structures

Incidence sequence for each vertex
sequence of references to edge objects of incident edges

Augmented edge objects
references to associated positions in incidence sequences of end vertices





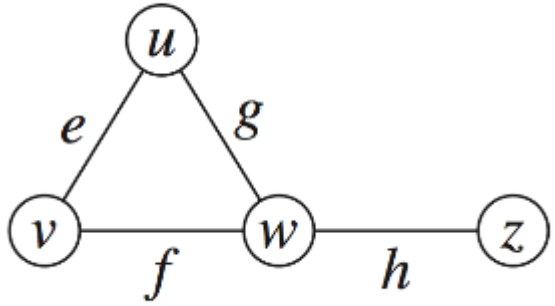
Adjacency Matrix Structure

Data Structures

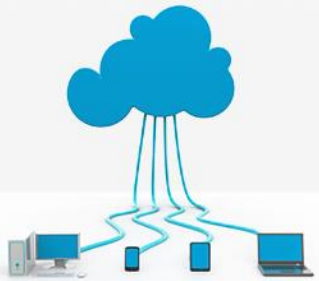
Edge list structure
Augmented vertex objects
    Integer key (index) associated with vertex

2D-array adjacency array
    Reference to edge object for adjacent vertices
    Null for non nonadjacent vertices

The “old fashioned” version
  just has 0 for no edge and 1 for edge

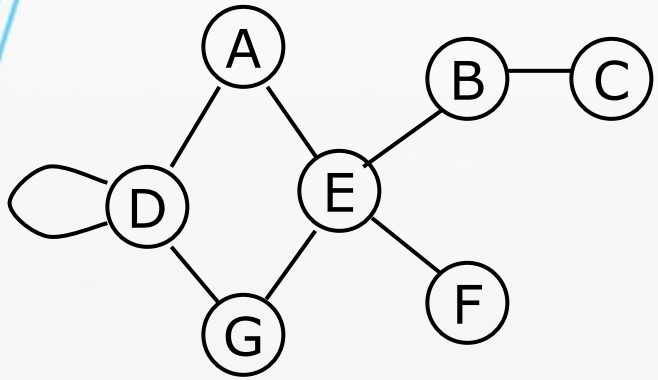


	0	1	2	3
u → 0		e	g	
v → 1	e		f	
w → 2	g	f		h
z → 3			h	



Graph

Adjacency-matrix representation (old fashioned)



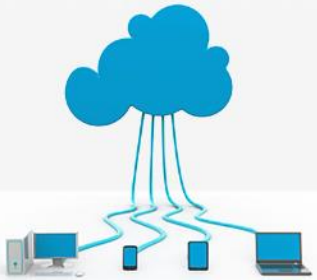
	A	B	C	D	E	F	G
A				●	●		
B			●		●		
C		●					
D	●			●			●
E	●	●				●	●
F					●		
G				●	●		

One simple way of representing a graph is the adjacency matrix

A 2-D array has a mark at [i][j] if there is an edge from node I to node j

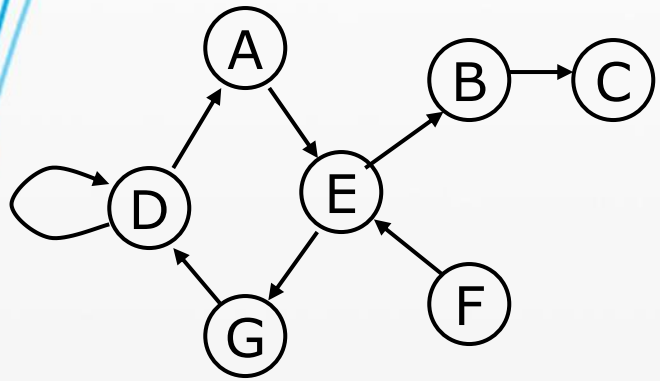
The adjacency matrix is symmetric about the main diagonal

This representation is only suitable for *small* graphs!



Graph

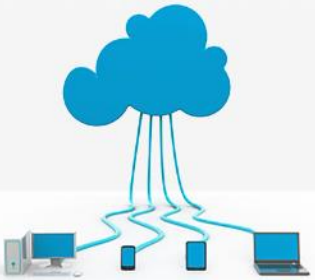
Adjacency-matrix representation (old fashioned)



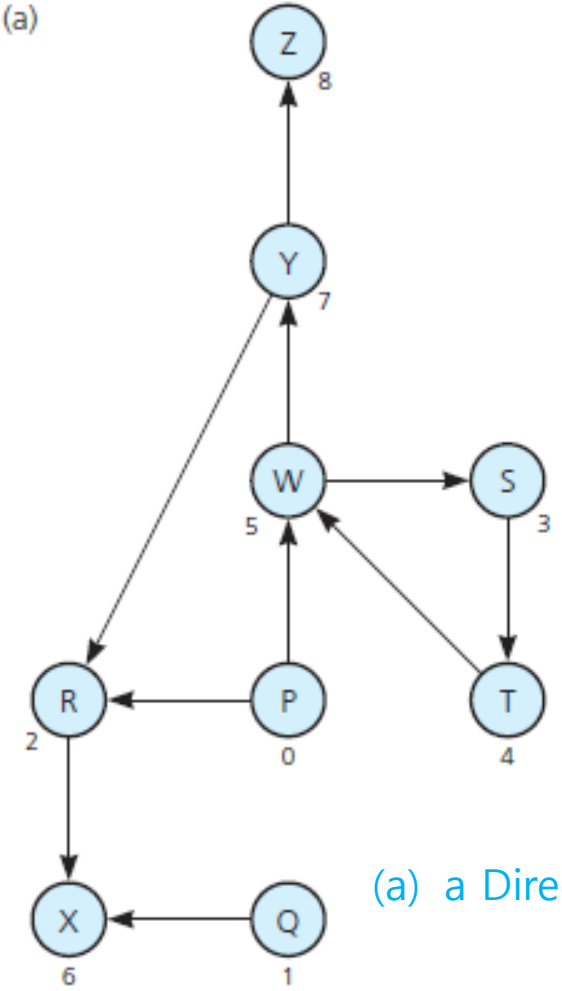
	A	B	C	D	E	F	G
A					●		
B			●				
C							
D	●			●			
E		●					●
F					●		
G				●			

An adjacency matrix can equally well be used for digraphs (directed graphs)

A 2-D array has a mark at [i][j] if there is an edge from node i to node j



(a)

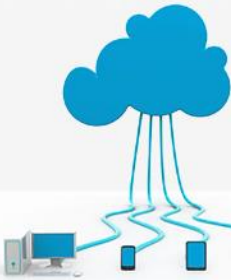


(a) a Direct Graph

(b)

		0	1	2	3	4	5	6	7	8
		P	Q	R	S	T	W	X	Y	Z
0	P	0	0	1	0	0	1	0	0	0
1	Q	0	0	0	0	0	0	1	0	0
2	R	0	0	0	0	0	0	1	0	0
3	S	0	0	0	0	1	0	0	0	0
4	T	0	0	0	0	0	1	0	0	0
5	W	0	0	0	1	0	0	0	1	0
6	X	0	0	0	0	0	0	0	0	0
7	Y	0	0	1	0	0	0	0	0	1
8	Z	0	0	0	0	0	0	0	0	0

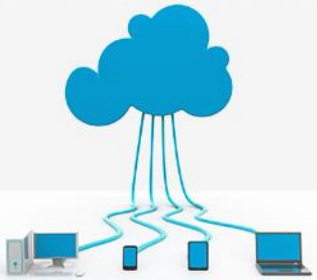
(b) its adjacency matrix



Graph

Graph Traversals

- ✓ is a systematic procedure for exploring a graph by examining all of its vertices and edges.
- ✓ is efficient if it visits all the vertices and edges in time proportional to their number, that is, in linear time.
- ✓ A key to answer questions involving **reachability**

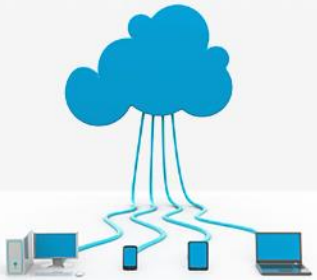


Graph

Graph Traversals

Some problems that deal with reachability in an **undirected graph** G:

- ✓ Computing a path from vertex u to vertex v, or reporting that no such path exists.
- ✓ Given a start vertex s of G, computing, for every vertex v of G, a path with the minimum number of edges between s and v, or reporting that no such path exists.
- ✓ Testing whether G is connected.
- ✓ Computing the connected components of G.
- ✓ Identifying a cycle in G, or reporting that G has no cycles.

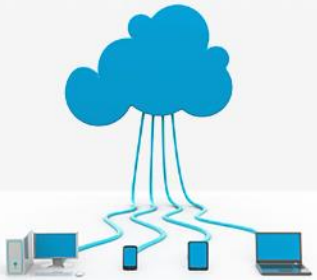


Graph

Graph Traversals

Some problems that deal with reachability in a **directed graph** G:

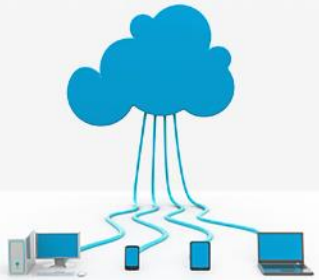
- ✓ Computing a directed path from vertex *u* to vertex *v*, or reporting that no such path exists.
- ✓ Finding all the vertices of *G* that are reachable from a given vertex *s*
- ✓ Determine whether *G* is acyclic.
- ✓ Determine whether *G* is strongly connected.



Graph

Graph Traversals

- ✓ Depth-First Search
  - Goes as far as possible from a vertex before backing up
  - Implemented
    - Recursive algorithm or
    - Iterative algorithm, using a stack
- ✓ Breadth-First Search
  - Visits all vertices adjacent to vertex before going forward
  - Breadth-first search uses a queue



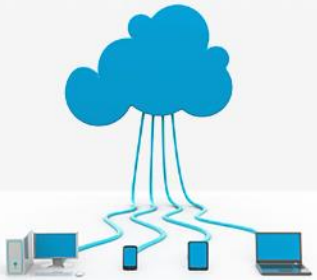
Graph Traversals

Depth-First Search

- ✓ The size depends both on
  - The number of edges, and
  - The out-degree of the vertices

Consider implementing a depth-first traversal on a graph:

- ✓ Choose any vertex, mark it as visited
- ✓ From that vertex:
  - If there is another adjacent vertex not yet visited, go to it
  - Otherwise, go back to the most previous vertex that has not yet had all of its adjacent vertices visited and continue from there
- ✓ Continue until no visited vertices have unvisited adjacent vertices



Graph Traversals

Breadth-First Search

Implementing a breadth-first traversal on a graph:

- ✓ Choose any vertex, mark it as visited and push it onto queue
- ✓ While the queue is not empty
  - Pop to top vertex v from the queue
  - For each vertex adjacent to v that has not been visited
    - Mark it visited, and
    - Push it onto the queue
- ✓ This continues until the queue is empty
  - Note: if there are no unvisited vertices, the graph is connected