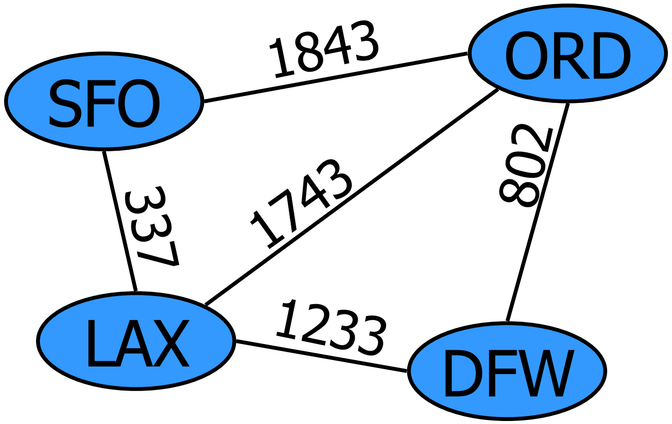
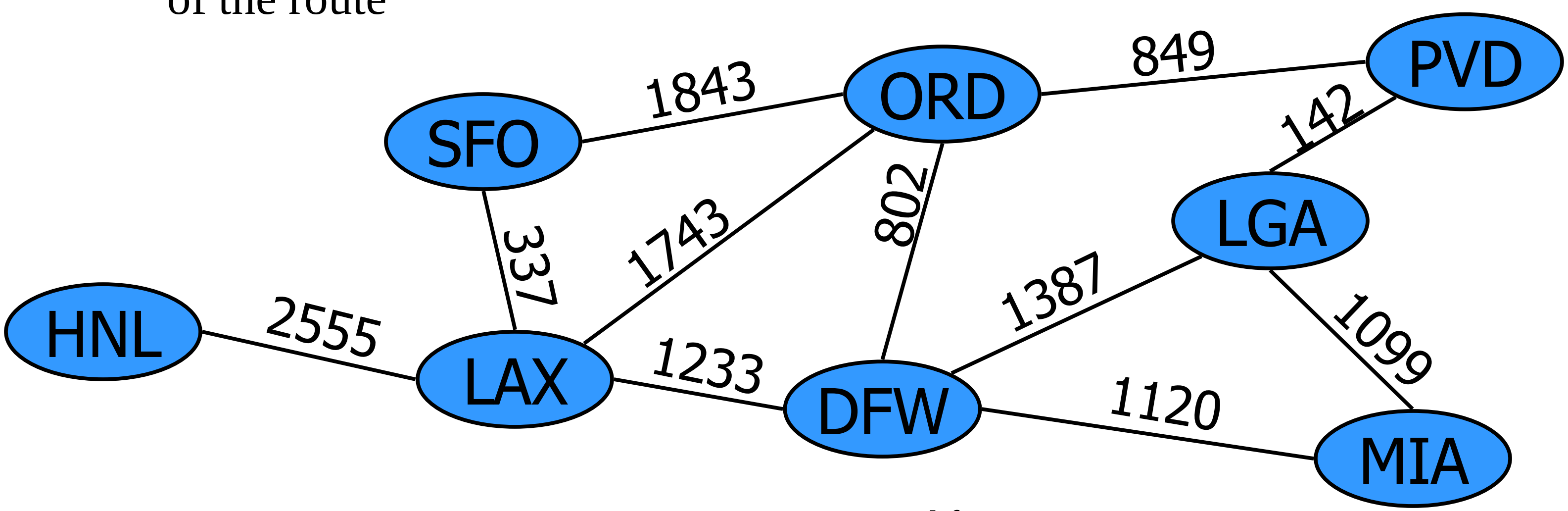
Graphs



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
  - Vertices and edges are positions and store elements
- Example:
  - A vertex represents an airport and stores the three-letter airport code
  - An edge represents a flight route between two airports and stores the mileage of the route

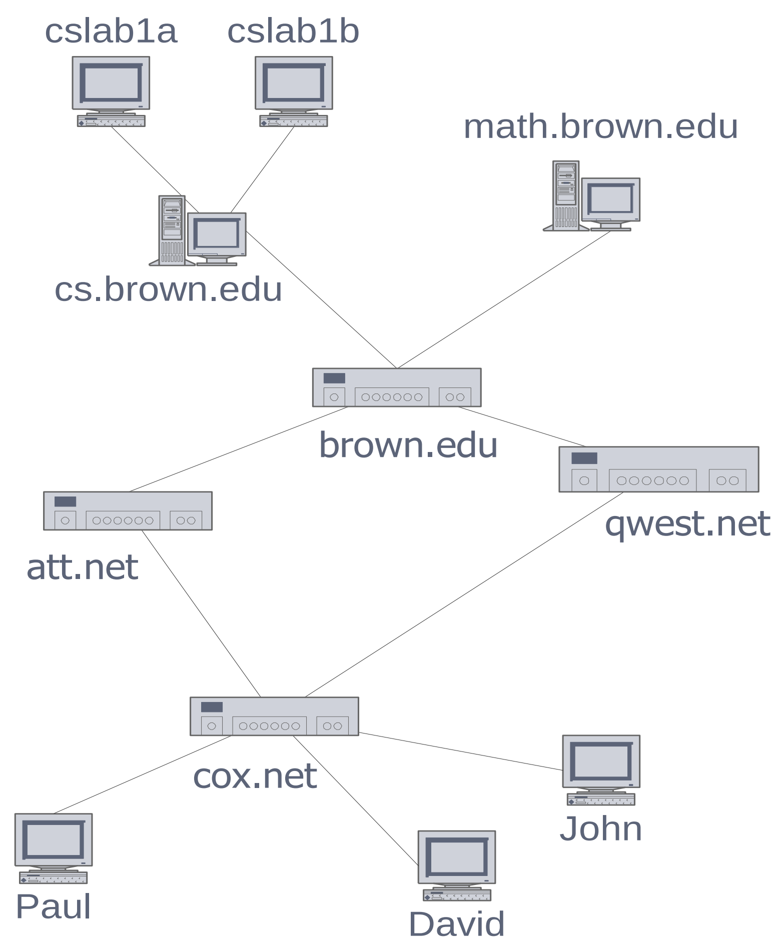


Edge Types

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

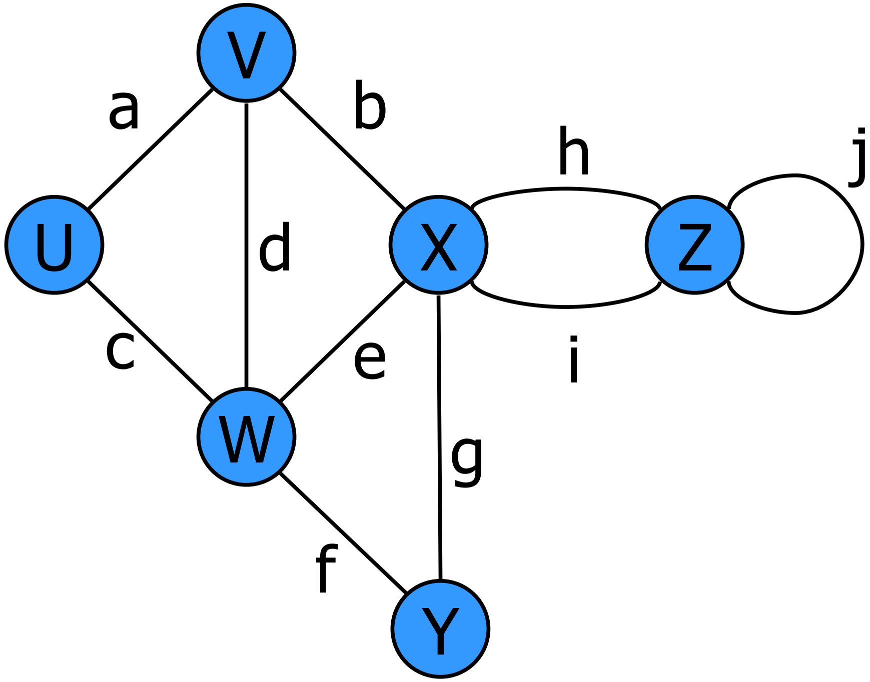
Applications

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



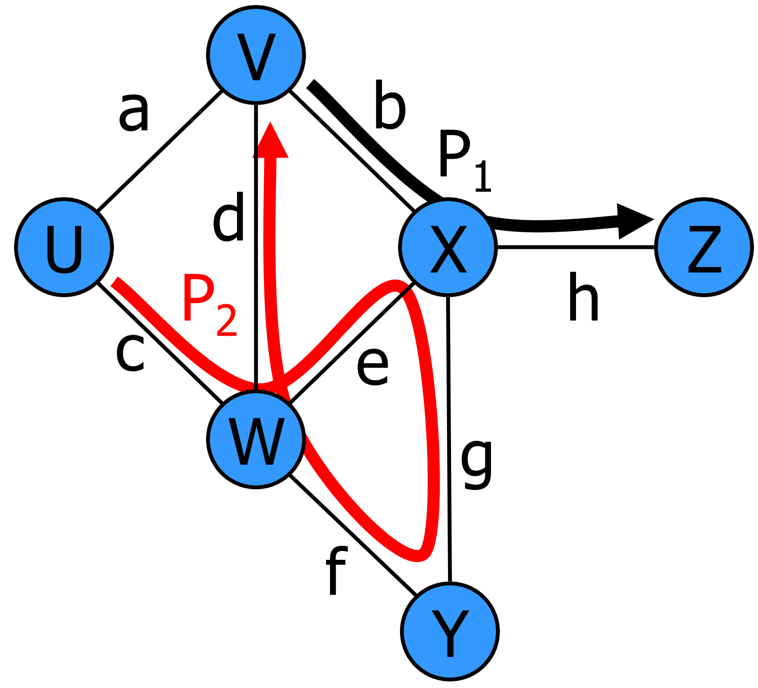
Terminology

- Vertices (or endpoints) of an edge
  - U and V are the *endpoints*
- Edges incident on a vertex
  - a, d, and b are *incident* on V
- Adjacent vertices
  - U and V are *adjacent*
  - *Having direct edge*
- Parallel edges
  - h and i are *parallel edges*
- Self-loop
  - j is a *self-loop*
- Degree of a vertex
  - X has *degree* 5
- In Degree
  - Number of edges moving to the vertex
- Out Degree
  - Number of edges moving out of the vertex



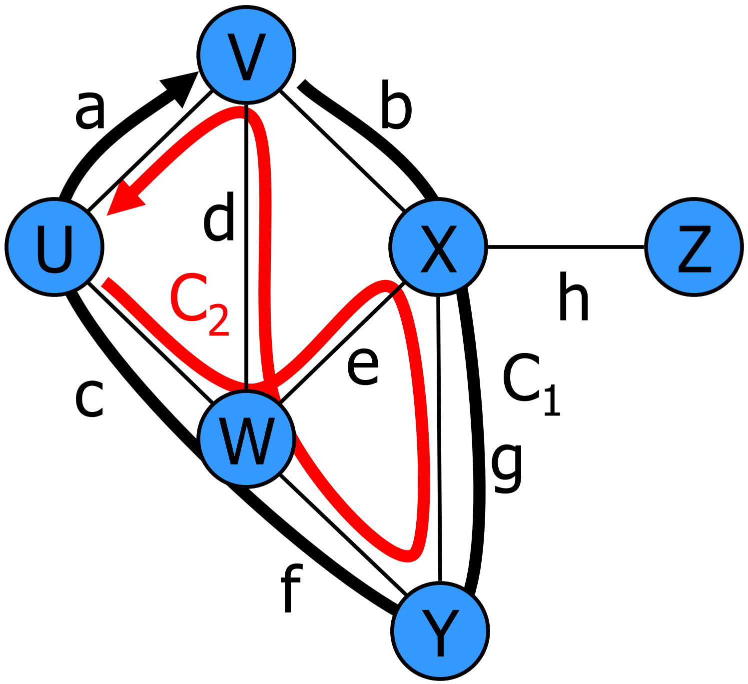
Terminology (cont.)

- Path
  - sequence of alternating adjacent vertices
  - begins with a vertex
  - ends with a vertex
- Simple path
  - path such that all its vertices are distinct
- Examples
  - P_1=(V,X,Z) is a simple path
  - P_2=(U,W,X,Y,W,V) is a path that is not simple



Terminology (cont.)

- Cycle
  - circular sequence of adjacent vertices
- Simple cycle
  - cycle such that all its vertices are distinct except the first one
- Examples
  - C_1=(V,X,Y,W,U,↵) is a simple cycle
  - C_2=(U, W, X,Y, W, V, ↵) is a cycle that is not simple



Graphs

General graphs differ from trees

- need not have a root node
- no implicit parent-child relationship
- may be several (or no) paths from one vertex to another.

Directed graphs *(direction associated with links)* are useful in modeling

- communication networks

  networks in which signals, electrical pulses, etc. flow from one node to another along various paths.

In networks where there may be no direction associated with the links,

- model using *undirected graphs*, or simply *graphs*.

Graph Functionality

Basic Graph Operations include

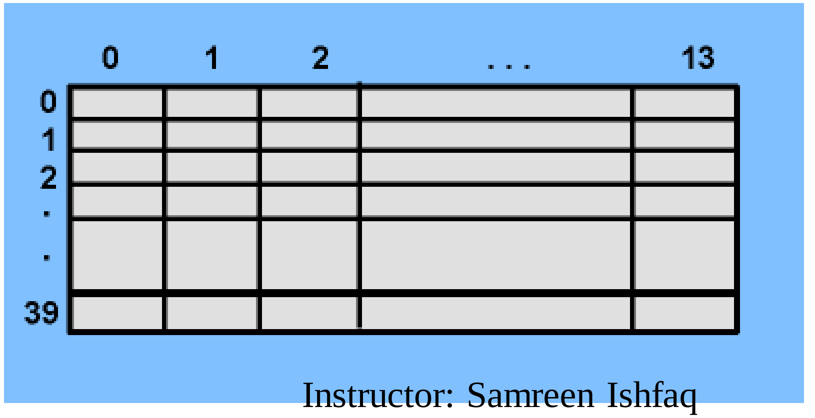
- Construct graph
- Check if it is empty
- Destroy a directed graph
- Insert a new node
- Insert directed edge between two nodes or from a node to itself
- Delete a node and all directed edges to or from it
- Delete a directed edge between two existing nodes
- Search for a value in a node, starting from a given node
- Traversal
- Determining if node *x* is reachable from node *y*
- Determining the number of paths from node *x* to node *y*
- Determining the shortest path from node *x* to node *y*

Multidimensional Array

- Two dimensional Array
  - used to store information that we normally represent in table form
  - Two-dimensional arrays, like one-dimensional arrays, are homogeneous
  - Examples of applications involving two-dimensional arrays include
    - a seating plan for a room (organized by rows and columns), a monthly budget (organized by category and month)

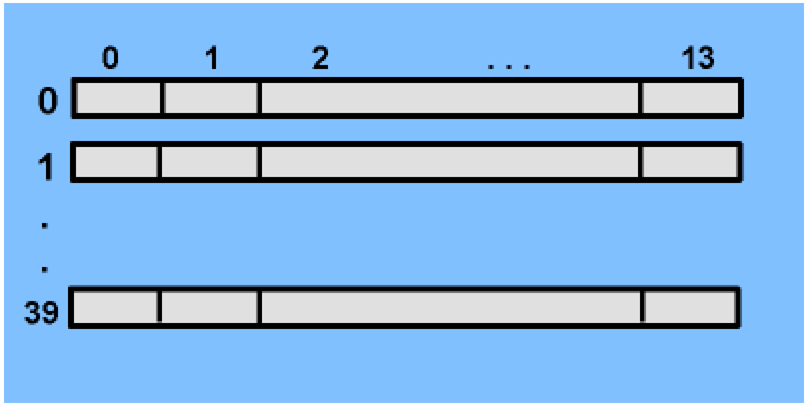
Declaration of Two-Dimensional Arrays

- const int MAX_STUDENTS=40;
- const int MAX_LABS=14;
- int labScores [MAX_STUDENTS][MAX_LABS];
- Manipulation of a two-dimensional array requires the manipulation of two indices



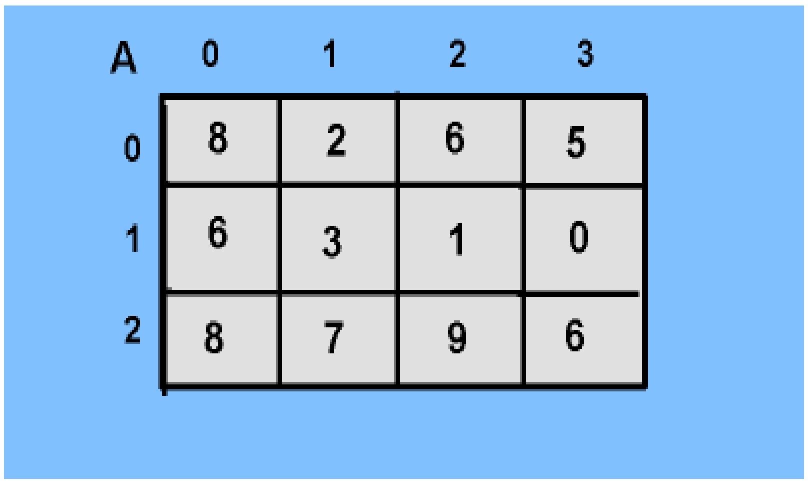
Two-Dimensional Arrays

- two-dimensional array may also be visualized as a one-dimensional array of arrays



Two-Dimensional Arrays

- ***Two-Dimensional Array Initialization***
  - int A[3][4] = {{8, 2, 6, 5}, //row 0
  - {6, 3, 1 ,0}, //row 1
  - {8, 7, 9, 6}}; //row 2
- ***Accessing a Two-Dimensional Array Element***
  - *A [2][3] = 6;*



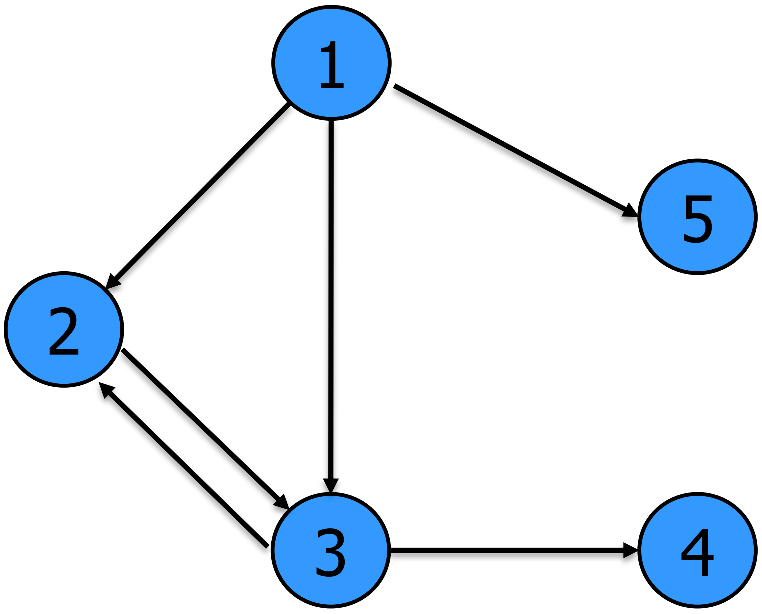
Graph Representation

Adjacency matrix representation

- For directed graph with vertices numbered 1, 2, . . . , *n* is the *n* **X** *n* matrix *adj*,
- In which the entry in row *i* and column *j* is 1 (or true) if vertex *j* is **adjacent** to vertex *i* (that is, if there is a directed arc from vertex *i* to vertex *j*), and is 0 (or false) otherwise.

Graph Representation

	1	2	3	4	5
1	0	1	1	0	1
2	0	0	1	0	0
3	0	1	0	1	0
4	0	0	0	0	0
5	0	0	0	0	0



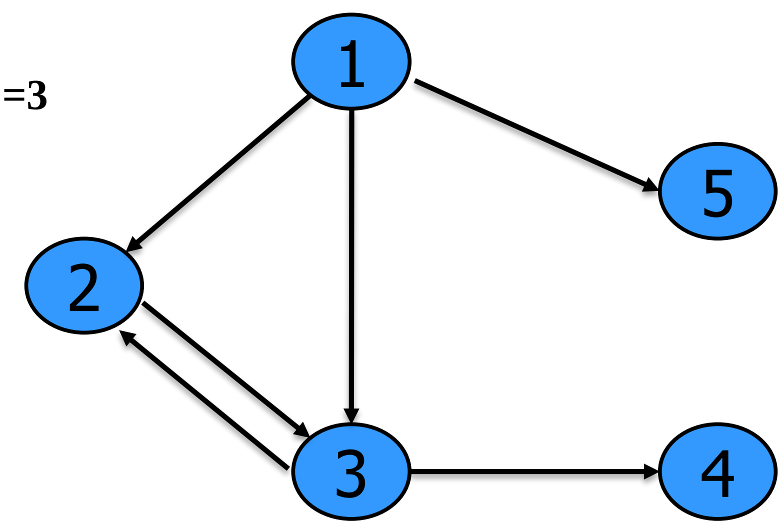
Adjacency Matrix

- The sum of 1's (or trues) in row *i* of the adjacency matrix is yields the *out-degree* of the *i*th vertex, i.e how many outgoing arcs emanate from i
- Similarly, the sum of the entries in the *i*th column is its *in-degree*.

0	1	1	0	1
0	0	1	0	0
0	1	0	1	0
0	0	0	0	0
0	0	0	0	0

outdegree(1) =3

indegree(3) =2



Adjacency Matrix

- The matrix representation of a digraph is also useful for path counting
- For example, a path of length 2 from vertex i to vertex j in a digraph G exists if there is some vertex k such that there is an arc from vertex i to vertex k and an arc from vertex k to vertex j.
- Hence both the i, k entry and the k, j entry of the adjacency matrix of G must be 1.

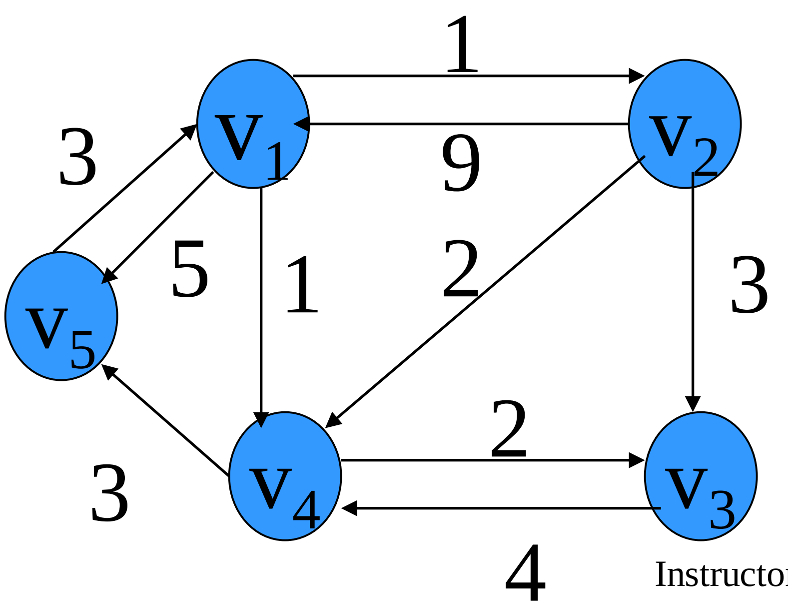
Graph Representation

For a **weighted digraph**

there is some “cost” or “weight” is associated with each arc

the cost of the arc from vertex *i* to vertex *j* is used instead of 1 in the adjacency matrix.

Clearly, some non-existent cost (such as 0, or –1, or infinity) must then be used to indicate when an edge does not exist.



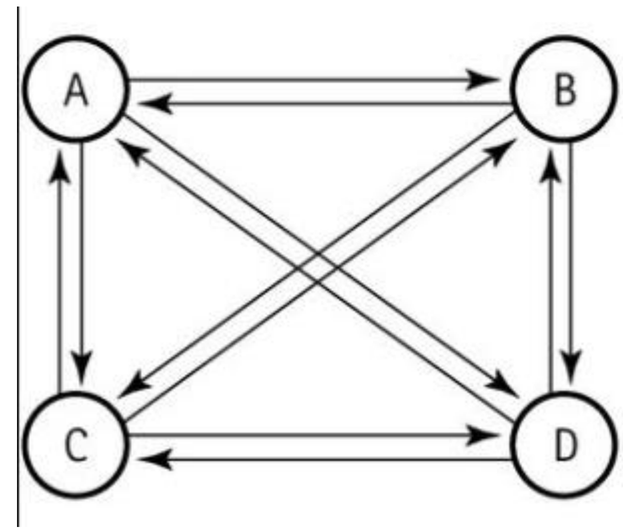
	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	∞
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	∞	∞	∞	0
		W			

Complete Graph

A complete graph is a graph in which there is an edge between each pair of vertices.

A digraph of n nodes has *n*(n-1)* arcs (edges).

n = 4
edges = 4*(4-1)
edges = 4 * 3
edges = 12



Adjacency Link List

- If graph is to be represented in Link list it must have two types of nodes
  - Vertex node
    - Information of vertex
    - Link to next vertex (Address of next vertex node)
    - Link to edge list(Address of edge list)
  - Edge Node
    - Link to vertex node(Address of vertex node)
    - Link to next edge node(Address of edge node)

Adjacency Link List