



**BITS Pilani**  
Hyderabad Campus

# Data Structures and Algorithms Design

Febin.A.Vahab  
2019-20

# CONTACT SESSION 6 -PLAN



Contact Sessions(#)	List of Topic Title	Text/Ref Book/external resource
6	Graphs - Terms and Definitions, Properties, Representations (Edge List, Adjacency list, Adjacency Matrix), Graph Traversals (Depth First and Breadth First Search )	T1: 6.1, 6.2, 6.3

# Graphs

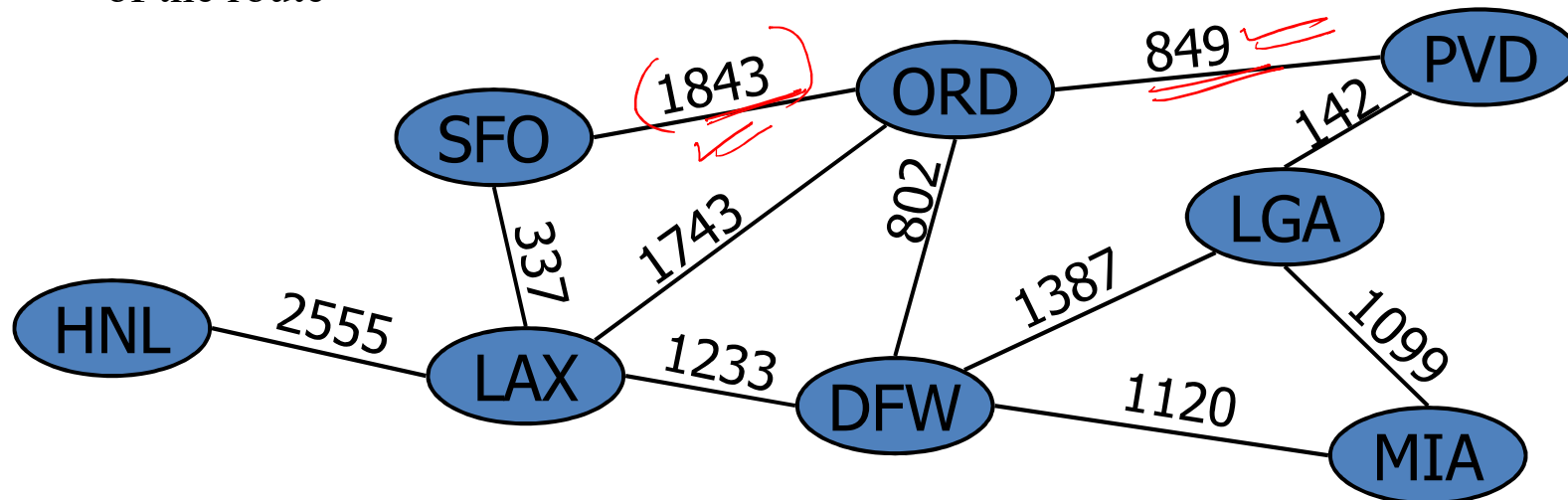


- Graphs
  - Definition
  - Applications
  - Terminology
  - Properties
  - ADT
- Data structures for graphs
  - Edge list structure
  - Adjacency list structure
  - Adjacency matrix structure

# Graphs



- 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



# Graphs



- 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

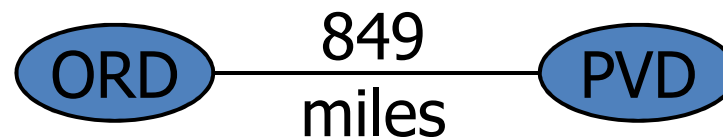
- unordered pair of vertices  $(u, v)$
- e.g., a flight route

- Directed graph

- all the edges are directed
- e.g., flight network

- Undirected graph

- all the edges are undirected
- e.g., route network



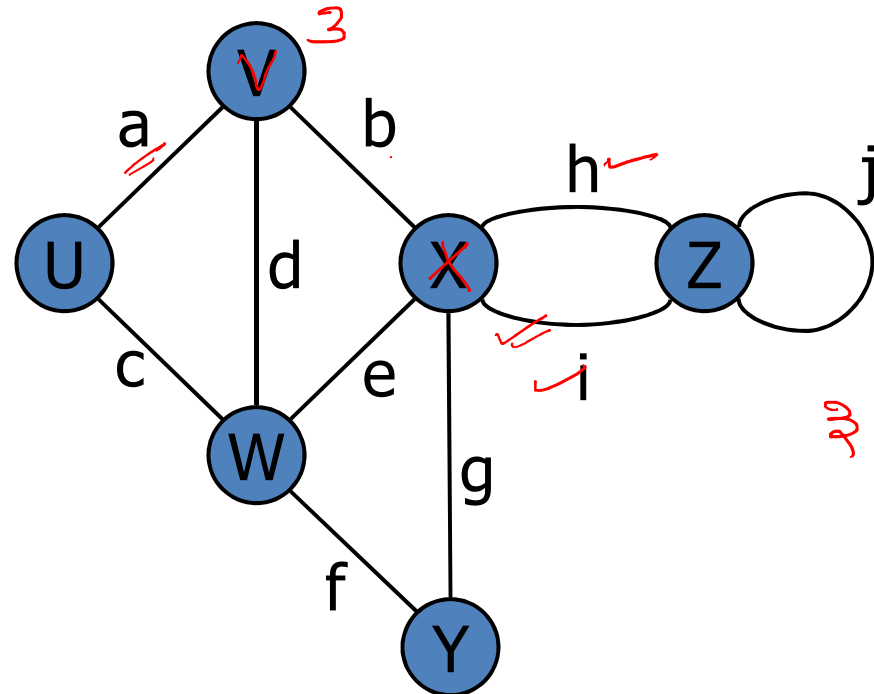


# Graphs-Applications

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

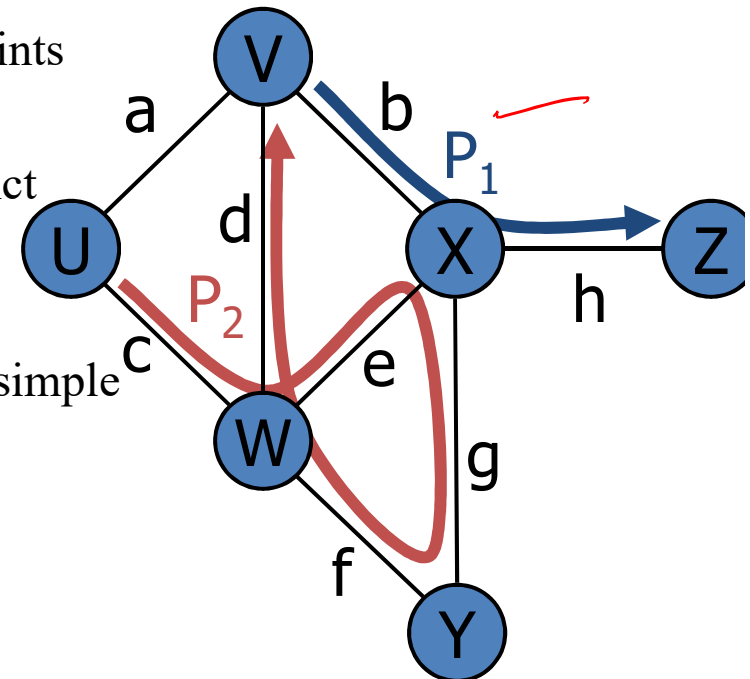
# Graphs-Terminology

- End vertices (or endpoints) of an edge
  - U and V are the endpoints of a
- ✓ Edges incident on a vertex
  - a, d, and b are incident on 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



# Graphs-Terminology

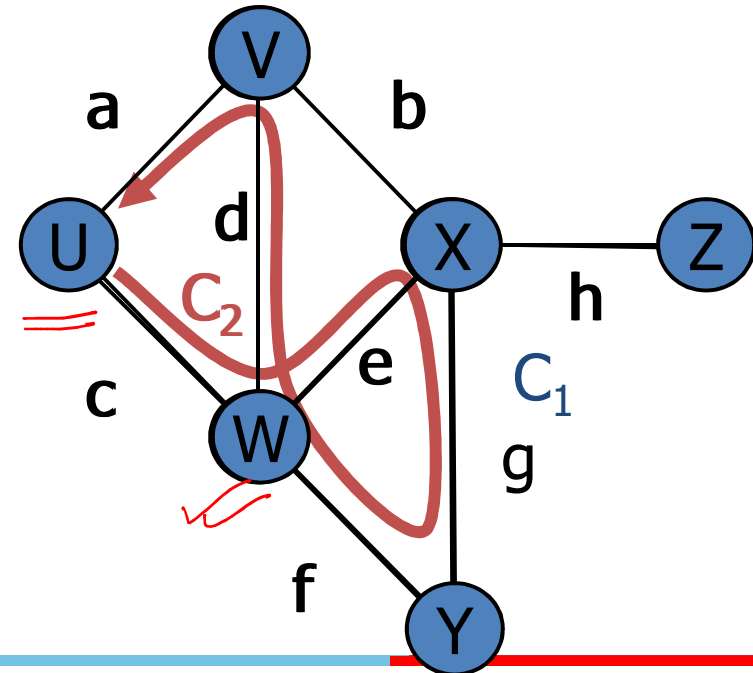
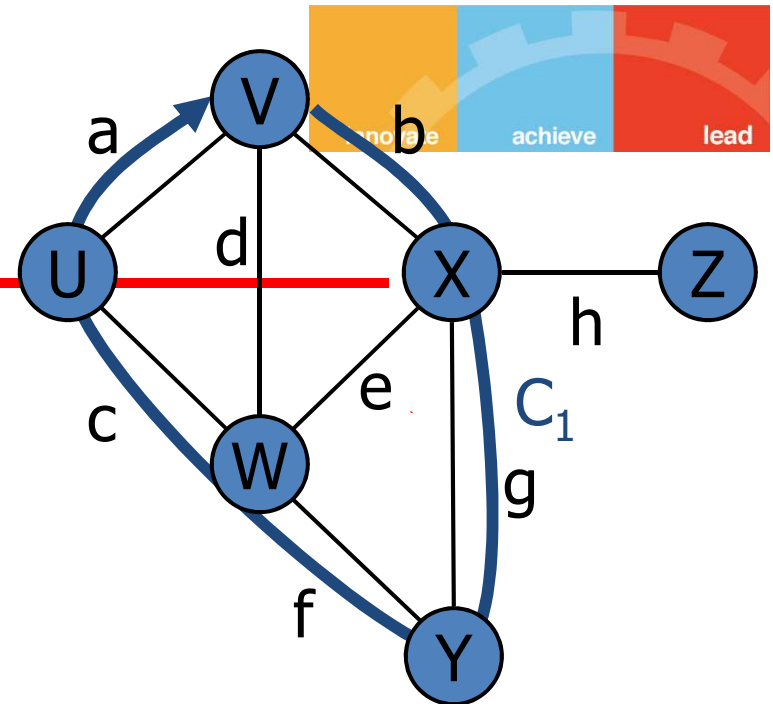
- Path
  - sequence of alternating vertices and edges ✓
  - begins with a vertex } ✓
  - ends with a vertex }
  - each edge is preceded and followed by its endpoints
- Simple path
  - path such that all its vertices and edges are distinct ✓ ✓
- Examples
  - $P_1 = (V, b, X, h, Z)$  is a simple path
  - $P_2 = (U, c, W, e, X, g, Y, f, W, d, V)$  is a path that is not simple





# Graphs-Terminology

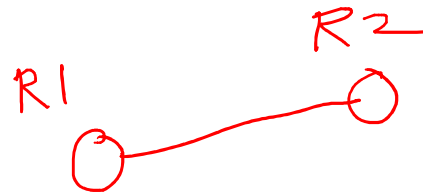
- Cycle
  - circular sequence of alternating vertices and edges
  - each edge is preceded and followed by its endpoints
- Simple cycle
  - cycle such that all its vertices and edges are distinct
- Examples
  - $C_1 = (V, b, X, g, Y, f, W, c, U, a)$  is a simple cycle
  - $C_2 = (U, c, W, e, X, g, Y, f, W, d, V, a)$  is a cycle that is not simple



# Graph-Example 1

We can visualize collaborations among the researchers of a certain discipline by constructing a graph whose vertices are associated with the researchers themselves, and whose edges connect pairs of vertices associated with researchers who have coauthored a paper or book.

Is it a directed or undirected graph?



Such edges are **undirected** because coauthorship is a symmetric relation; that is, if A has coauthored something with B, then B necessarily has coauthored something with A .



# Graph-Example 2

- We can associate with an object-oriented program a graph whose vertices represent the classes defined in the program, and whose edges indicate inheritance between classes. There is an edge from a vertex  $v$  to a vertex  $u$  if the class for  $v$  extends the class for  $u$ .
- Is it a directed or undirected graph?
- Such edges are directed because the inheritance relation only goes in one direction (that is, it is asymmetric).





# Graph-Example 3

- A city map can be modelled by a graph whose vertices are intersections or dead ends, and whose edges are stretches of streets without intersections.
- Directed or undirected?
- This graph has **both undirected edges**, which correspond to stretches of two-way streets, **and directed edges**, which correspond to stretches of one-way streets. Thus, a graph modelling a city map is a mixed graph



# Graph-Example 4

---

- Physical examples of graphs are present in the electrical wiring and plumbing networks of a building.
- Such networks can be modelled as graphs, where each connector, fixture, or outlet is viewed as a vertex, and each uninterrupted stretch of wire or pipe is viewed as an edge.
- Such graphs are actually components of much larger graphs, namely the local power and water distribution networks.
- Depending on the specific aspects of these graphs that we are interested in, we may consider their edges as undirected or directed, for, in principle, water can flow in a pipe and current can flow in a wire in either direction.



# Graph-Example 5-Path and Cycle

Given a graph  $G$  representing a city map, we can model a couple driving from their home to dinner at a recommended restaurant as traversing a path through  $G$ .

If they know the way, and don't accidentally go through the same intersection twice, then they traverse a simple path in  $G$ .

Likewise, we can model the entire trip the couple takes, from their home to the restaurant and back, as a cycle.

If they go home from the restaurant in a completely different way than how they went, not even going through the same intersection twice, then their entire round trip is a simple cycle. ✓✓

Finally, if they travel along one-way streets for their entire trip, then we can model their night out as a directed cycle.

# Graphs-Properties

## Property 1

If  $G$  is a graph with  $m$  edges, then

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

Proof: each edge is counted twice

## Property 2

Let  $G$  be a simple graph with  $n$  vertices and  $m$  edges.

In an undirected graph with no self-loops and no multiple edges

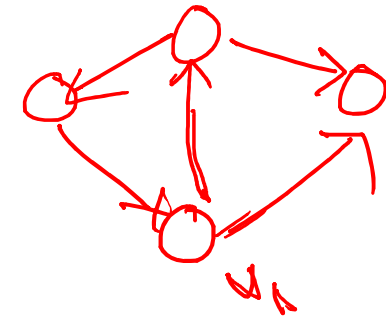
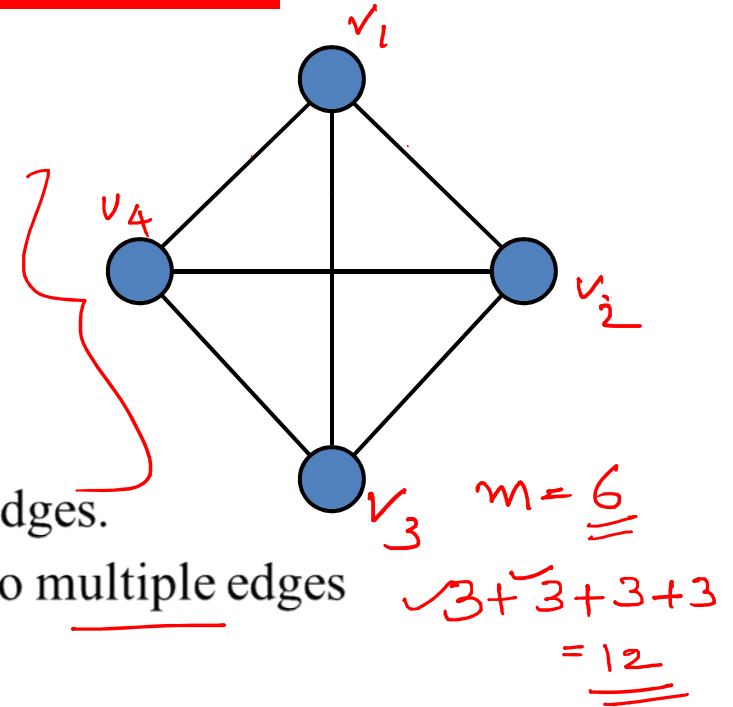
$$m \leq n(n-1)/2 \text{ is } O(n^2)$$

Proof: each vertex has degree at most  $(n-1)$

## Property 3

If  $G$  is a directed graph with  $m$  edges, then

$$\sum_{v \in G} \text{indeg}(v) = \sum_{v \in G} \text{outdeg}(v) = m$$

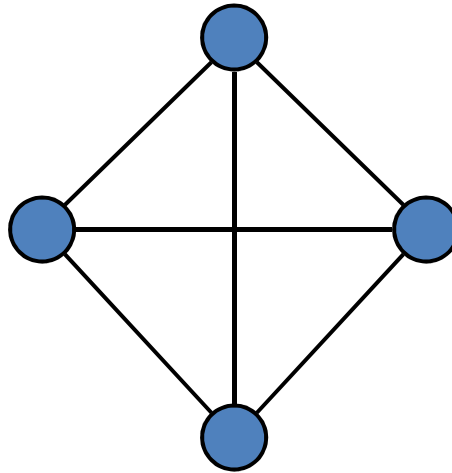


# Graphs-Properties



## Example

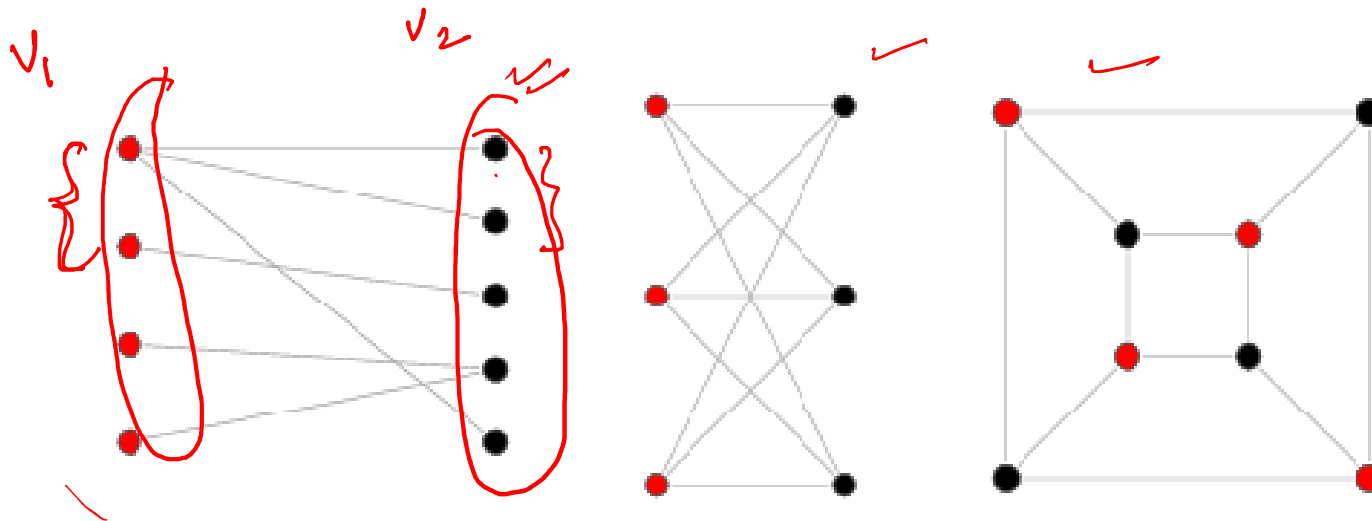
- $n = 4$
- $m = 6$
- $\deg(v) = 3$





# Bipartite Graph

- If the vertex-set of a graph  $G$  can be split into two disjoint sets,  $V_1$  and  $V_2$ , in such a way that each edge in the graph joins a vertex in  $V_1$  to a vertex in  $V_2$ , and there are no edges in  $G$  that connect two vertices in  $V_1$  or two vertices in  $V_2$ , then the graph  $G$  is called a bipartite graph.



# Applications



- **Document/Term Graphs:** Here U are documents and V are terms or words, and there is an edge  $(u, v)$  if the word v is in the document u. Such graphs are use often to analyze text, for example to cluster the documents.
- **Movies preferences:** The viewers are the vertices U and the movies the vertices V and there is an edge from u to v if u viewed v. In this case the edges are weighted by the rating the viewer gave.
- **Students and classes:** We might create a graph that maps every student to the classes they are taking. Such a graph can be used to determine conflicts, e.g. when classes cannot be scheduled together.

# Graphs-ADT

- A graph is a positional container of elements that are stored at the graph's vertices and edges
- Vertices and edges
  - are positions
  - store elements
- **Accessor methods**
  - incidentEdges(v): Return an iterator of the edges incident upon v ✓
  - endVertices(e): Return an array of size 2 storing the end vertices of e.
  - degree(v):
  - adjacentVertices(v): Return an iterator of the vertices adjacent to v.
  - opposite(v, e): Return the endpoint of edge e distinct from v.
  - areAdjacent(v, w): Return whether vertices v and w are adjacent



## Complexity

## Methods Dealing with Directed Edges

- directed Edges(): Return an iterator of all directed edges.
- undirected Edges(): Return an iterator of all undirected edges.
- destination( e): Return the destination of the directed edge e.
- origin (e): Return the origin of the directed edge e.
- isDirected(e): Return true if and only if the edge e is directed.

## Update methods

- insertVertex(o): Insert and return a new vertex storing the object o
- insertEdge(v, w, o): Insert and return an undirected edge between vertices v and w, storing the object o.
- insertDirectedEdge(v, w, o)
- removeVertex(v): Remove vertex v and all its incident edges.
- removeEdge(e)
- Generic methods
  - numVertices()
  - numEdges()
  - vertices(): Return an iterator of the vertices of G.
  - edges()

# Graphs-ADT



Also supports

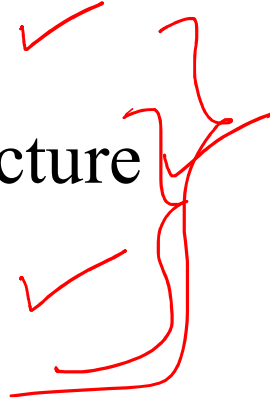
- `size()`
- `isEmpty ( )`
- `elements ( )`
- `positions()`
- `replaceElement(p, o )`
- `swapElements (p , q)`

where  $p$  and  $q$  denote positions, and  $o$  denotes an object (that is, an element)

# Data Structure for Graphs

---

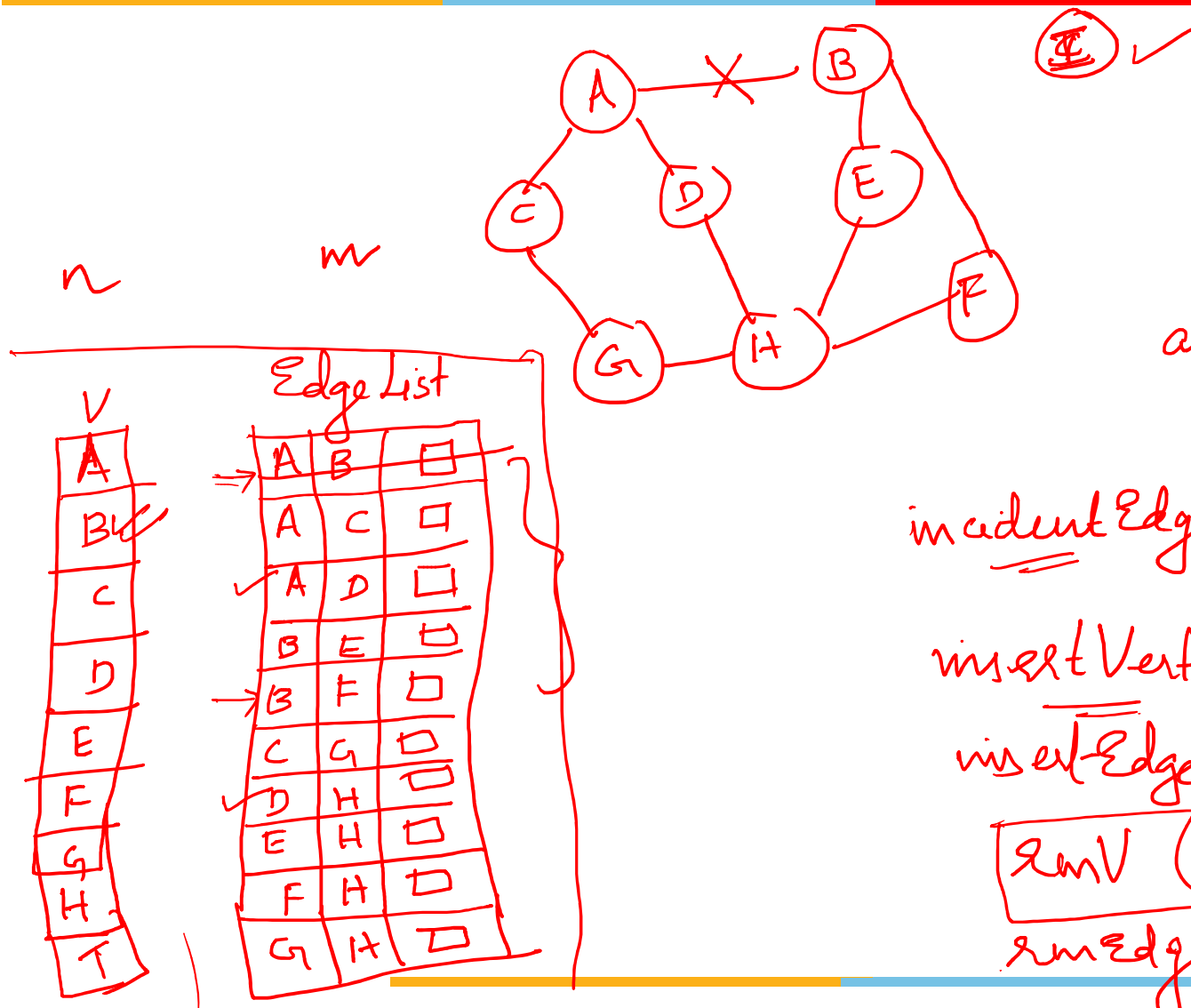
- Edge list structure ✓
- Adjacency list structure ✓
- Adjacency matrix ✓



# Edge List Structure

- Vertex object
  - Element, o
- Edge object
  - element
  - origin vertex object
  - destination vertex object
- Vertex sequence
  - sequence of vertex objects
- Edge sequence
  - sequence of edge objects





adjacent  $\left( \frac{v, w}{B, F} \right) = \underline{\underline{O(m)}}$

incident Edges  $(v) = \underline{\underline{O(m)}}$   
D

insert Vertex  $(o) = \underline{\underline{O(1)}}$

insert Edge  $(v, w, o) = \underline{\underline{O(1)}}$

rmV  $(v) = \underline{\underline{O(1)}}$

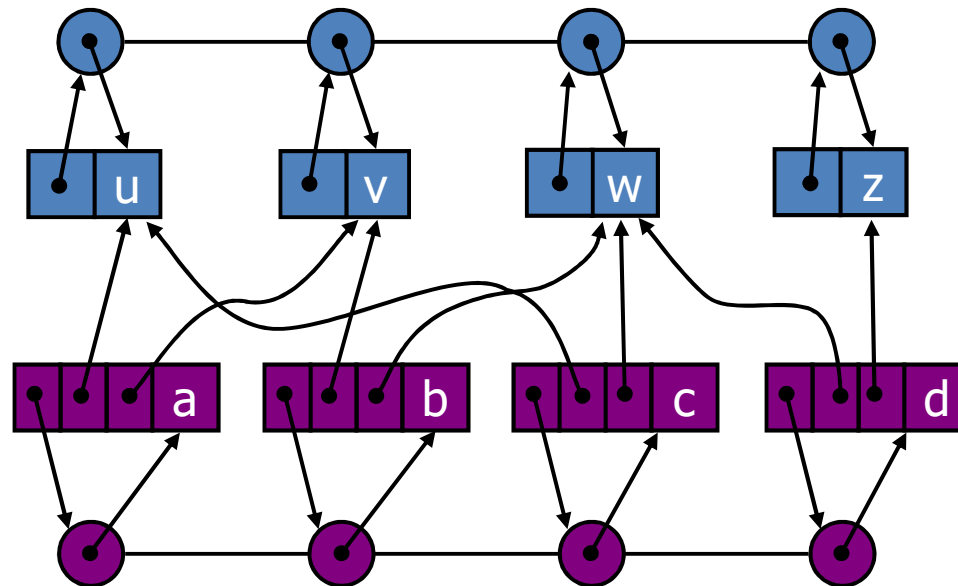
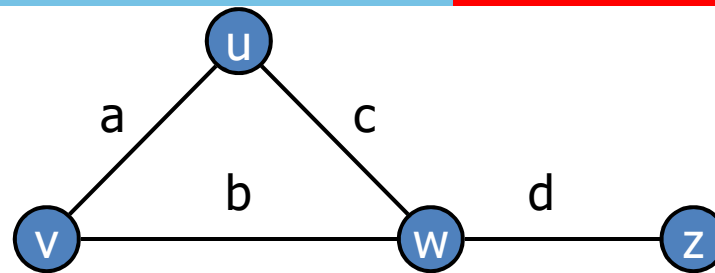
rmEdge  $(\text{edge}) = \underline{\underline{O(1)}}$

# Time Complexity



Methods	Edge List-Time Complexity
<code>incidentEdges(v)</code>	$O(m)$
<code>areAdjacent(v, w)</code>	$O(m)$
<code>insertVertex(o)</code>	$O(1)$
<code>insertEdge(v, w, o)</code>	$O(1)$
<code>removeVertex(v)</code>	$O(m)$
<code>removeEdge(e)</code>	$O(1)$

# Edge List Structure

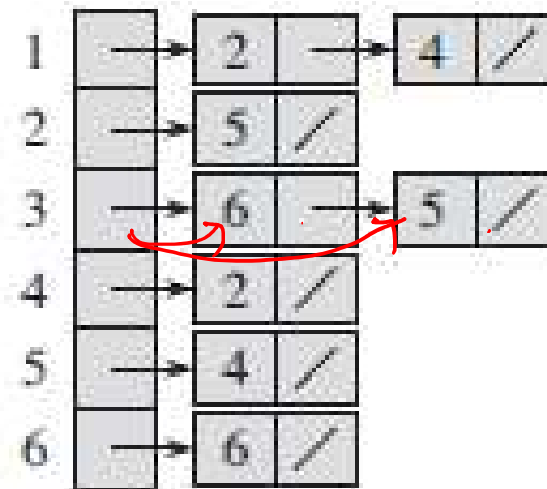
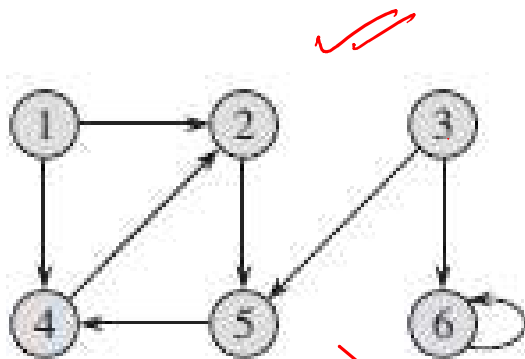
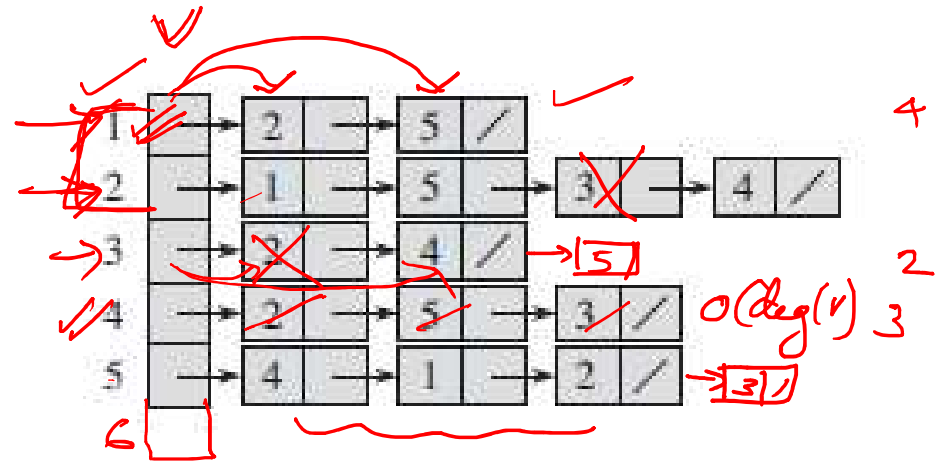
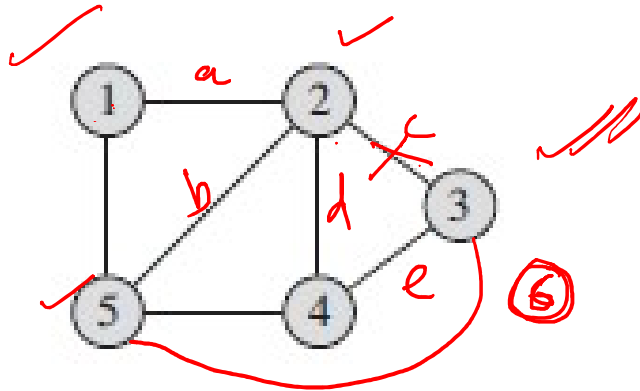





# Adjacency List Structure

- Extends edge list structure
- Add extra information that supports direct access to the incident edges (and thus to the adjacent vertices) of each vertex
- For each vertex  $v$ , store a reference to a list of the vertices adjacent to it.

# Adjacency List





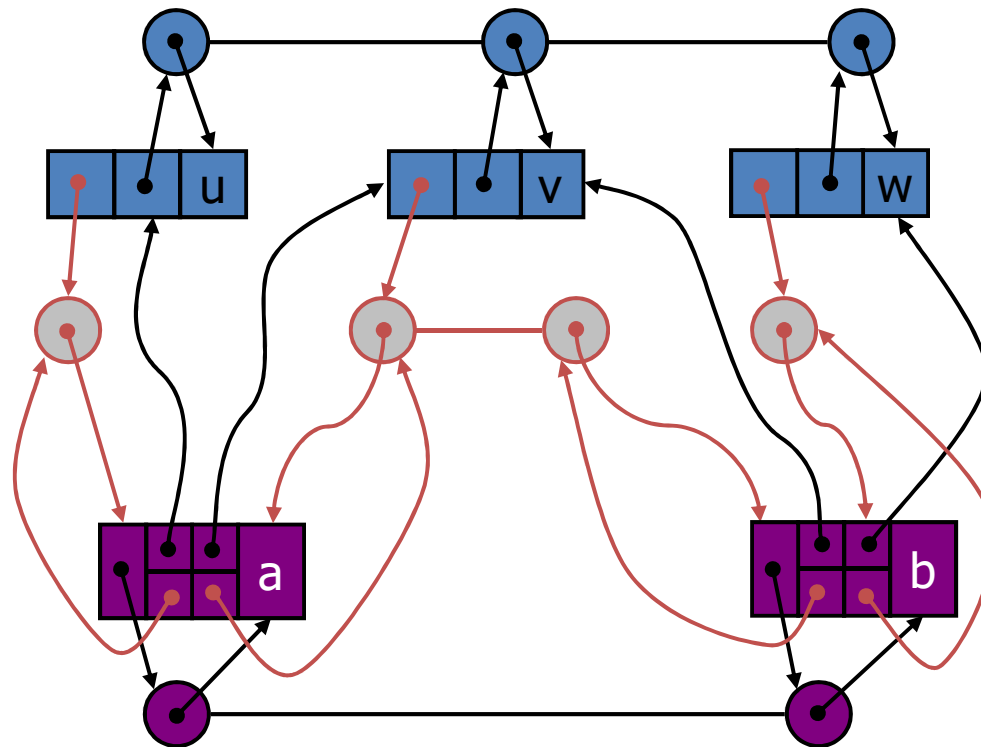
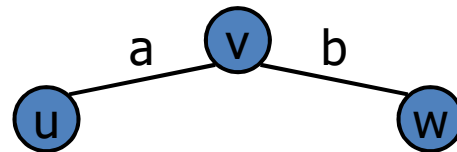
if  $\underline{v} = \underline{0}(\deg(v))$   
 $\text{ord}(\underline{v}, \underline{w}) = \underline{0}(\min(\deg v, \deg w))$

# Time Complexity



	Edge List	Adjacency List ✓✓
incidentEdges(v)	m	deg(v)
areAdjacent(v, w)	m	min(deg(v), deg(w))
insertVertex(o)	1	1
insertEdge(v, w, o)	1	1
removeVertex(v)	m	deg(v)
removeEdge(e)	1	1

# Adjacency List Structure





# Adjacency Matrix Structure

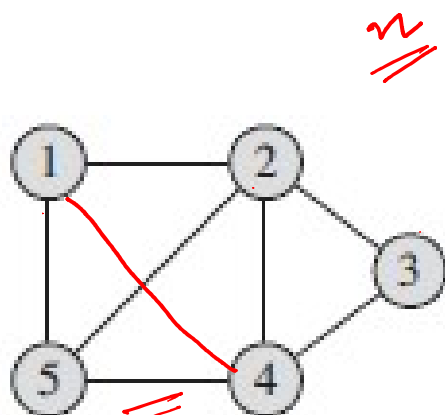
- The *adjacency-matrix representation* of a graph  $G = (V, E)$ , the vertices are numbered  $1, 2, \dots, |V|$  in some arbitrary manner. Then the adjacency-matrix representation of a graph  $G$  consists of a  $|V| \times |V|$  matrix

$A = (a_{ij})$  such that

$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$



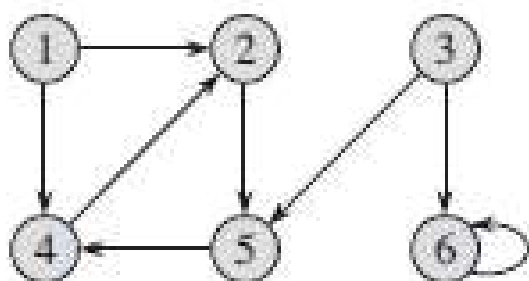
# Adjacency Matrix



$\text{incidentEdges}(v)$   
 $\text{areAdj}(2, 4)$   
 $\text{isVertex}(v)$   
 $\text{isEdge}(e)$   
 $\text{isVertex}(v)$   
 $\text{isEdge}(e)$

	1	2	3	4	5	6
1	0	1	0	1	1	
2	1	0	1	1	1	
3	0	1	0	1	0	
4	1	1	1	0	1	
5	1	1	0	1	0	

$O(n)$   
 $O(1)$   
 $O(n^2)$   
 $O(1)$   
 $O(n^2)$   
 $O(1)$



	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

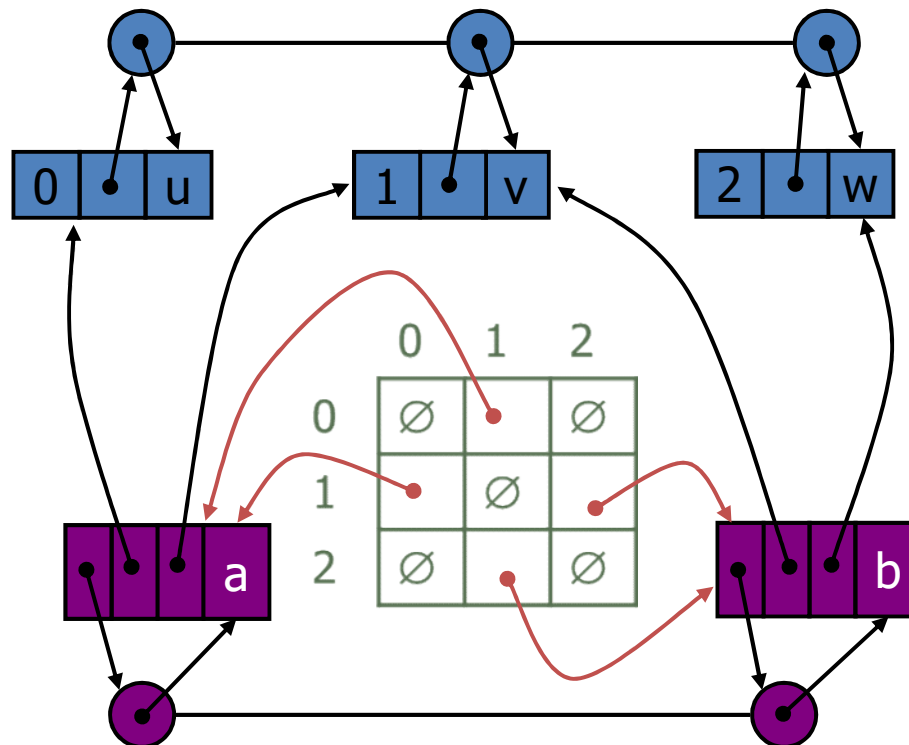
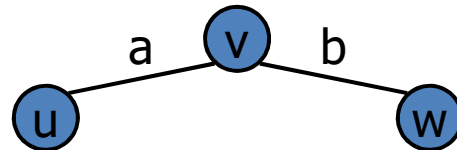


# Adjacency Matrix Structure

---

- Extends Edge list structure
- Augmented vertex objects
  - Integer key (index) associated with vertex
- 2D adjacency array
  - Reference to edge object for adjacent vertices
  - Null for nonadjacent vertices
- 0 for no edge and 1 for edge

# Adjacency Matrix Structure



# Asymptotic Performance

	Edge List ✓	Adjacency List ✓	Adjacency Matrix ✓
incidentEdges(v)	m	deg(v)	n
areAdjacent(v, w) ✓	m	<u>min(deg(v), deg(w))</u>	<u>1</u> ✓
insertVertex(o)	1	1	$n^2$
insertEdge(v, w, o)	1	1	1
removeVertex(v)	m	deg(v)	$n^2$
removeEdge(e)	1	1	1

Graph ADT



# THANK YOU!

**BITS Pilani**  
Hyderabad Campus