TECHNO INDIA UNIVERSITY
WESTBENGAL

Data Structures & Algorithms
(PCC-CS 301)

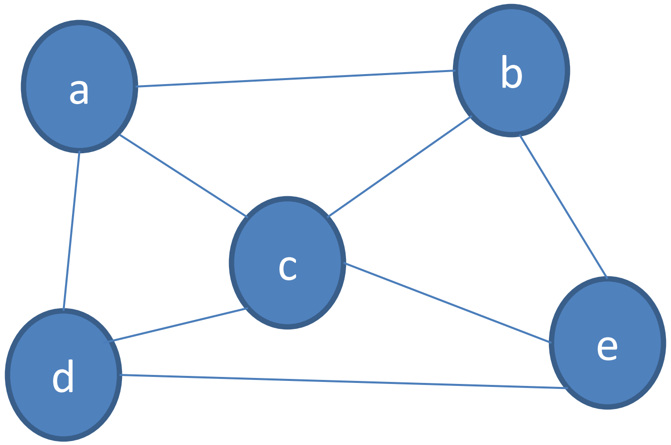
Dr. Debashis Das
Associate Professor
Department of CSE
Techno India University, Kolkata

Topics Covered

1. Graph Data Structure
    1.1. Introduction
    1.2. Types of Graph
    1.3. Memory representation
    1.4. Graph traversal algorithms

Graph: Introduction

- Graph: a data structure
  - The concept of Graph theory is used as an advanced level of structures for storing large data in memory and also for efficient accessing/ retrieving
  - Graph is a non-linear data structure like tree



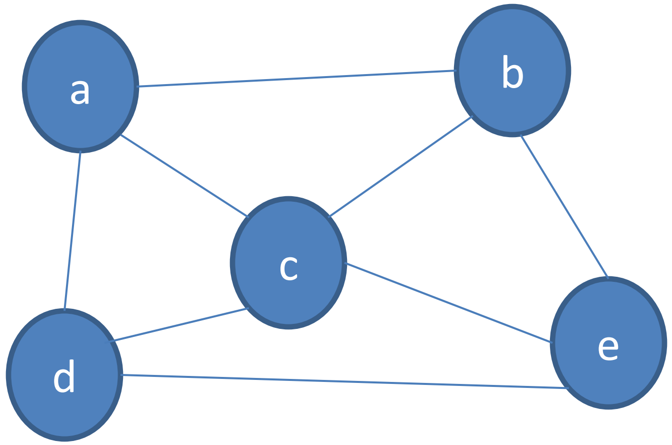
Graph: Introduction

- Definition

  A simple Graph is a pair *G=(V,E)* where *V* and *E* are finite sets and every element of *E* is two element subset of *V* (i.e. an unordered pair of distinct elements of *V*). The elements of *V* are called **vertices** and the elements of *E* are called **edges**

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

***E*={ab, ac, ad, bc, be, cd, ce, de}**

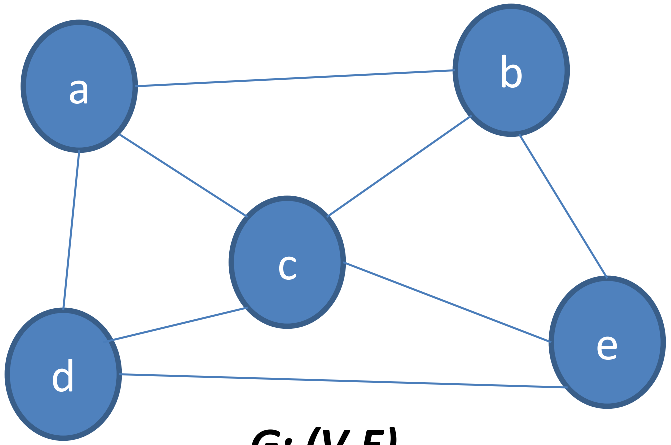


G: (V,E)

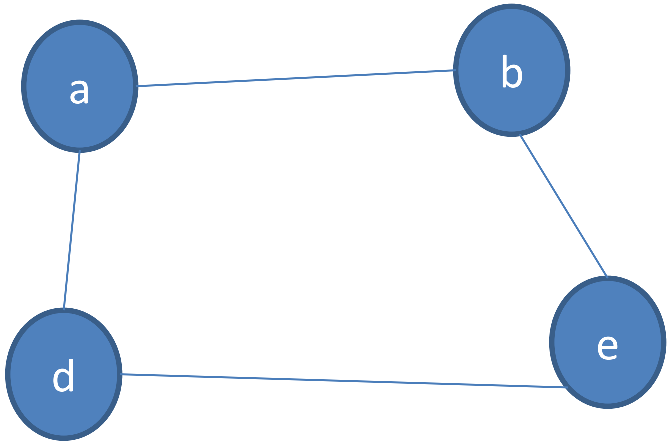
Graph: Introduction

- **Sub-graph**

  If *G=(V,E)* is a graph and *G'=(V',E')* where $V' \subseteq V$ and $E' \subseteq E$, *G'* is called the sub-graph of *G*



G: (V,E)



G': (V',E')

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

***E*={ab, ac, ad, bc, be, cd, ce, de}**

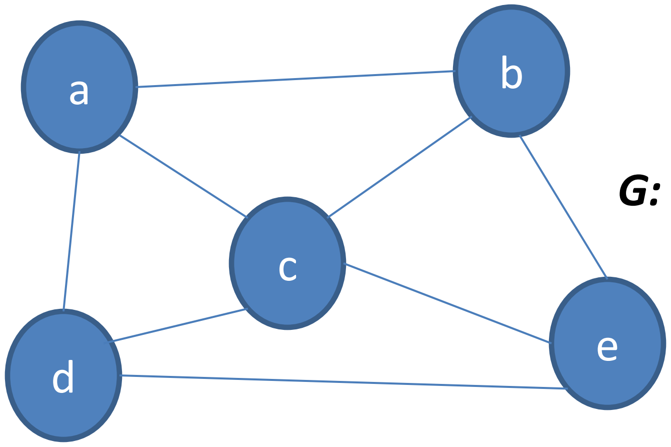
***V'*={a, b, d, e}**

***E'*={ab, ad, be, de}**

Graph: Introduction

- **Degree of vertex**

  The degree of a vertex is the number of edges that are incident on it (or connected to it)



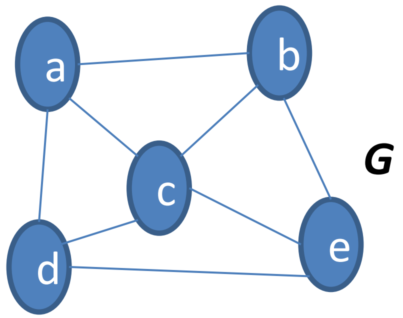
G: (V,E)

deg(a)=3	deg(b)=3	deg(c)=4	deg(d)=3	deg(e)=3

Graph: Introduction

- **Walk**

  A walk from any vertex ***s*** to ***t*** of a graph is the sequence of edges that are connecting ***s*** and ***t*** via a set of intermediate vertices.
  [Example: (ad,de,eb) is a walk from ***a*** to ***b*** of graph G]
  Length of a walk is the number of edges that form the walk



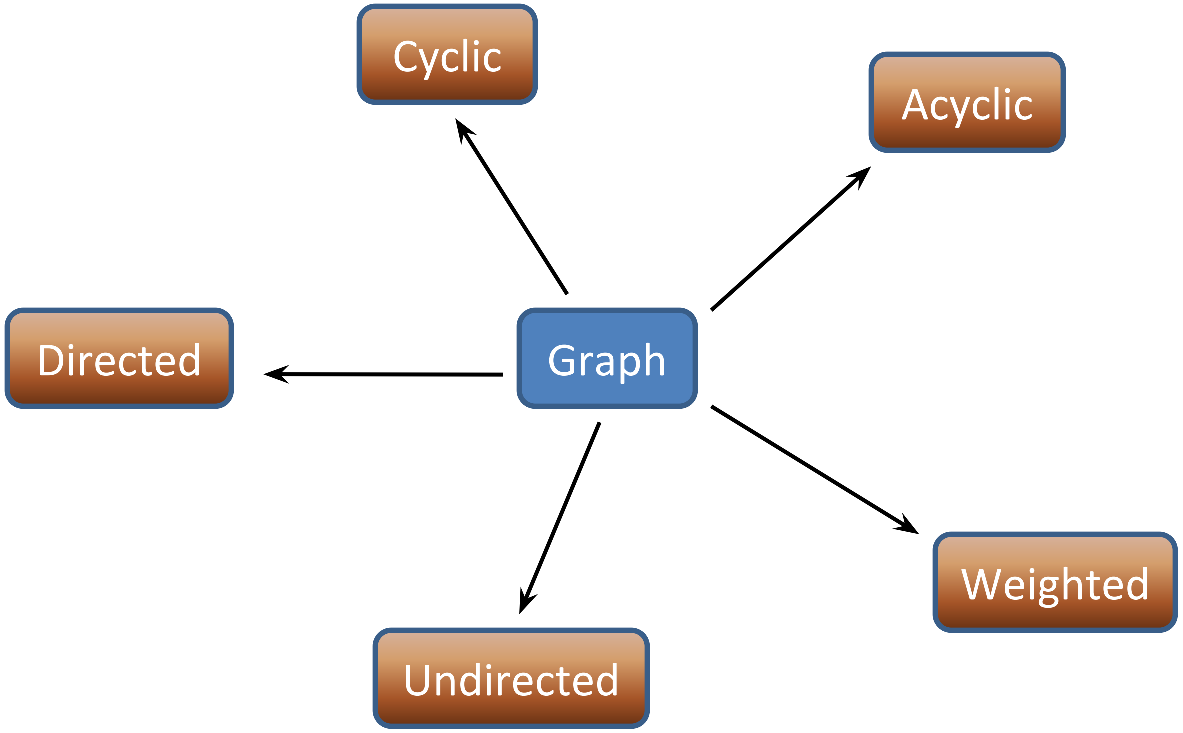
- **Path**

  A path is a walk whose vertices are all distinct.
  Path is represented through the vertices only. (adeb) is a path.
  (adcab) is a walk but not a path because ***a*** arises two times

- **Cycle**

  A cycle is a closed walk of length at least 3 whose interior vertices are all distinct. [Example: (adeca) is a cycle]

Graph: Types

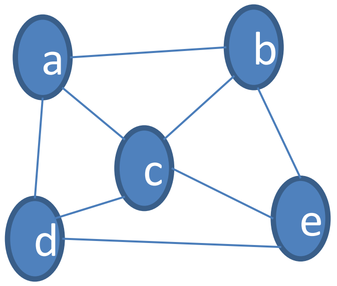


There are number of graphs exist. The mentioned types are useful for graph data structure

Graph: Types

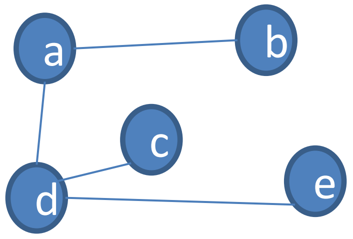
- **Cyclic graph**

  A graph is called cyclic if it contains at least one cycle



- **Acyclic graph**

  A graph contains no cycle is called acyclic graph



Tree is an acyclic graph

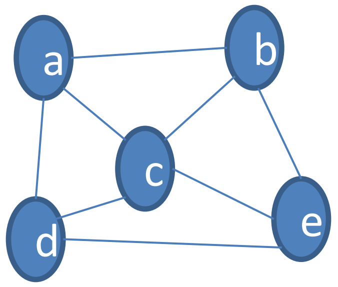
Graph: Types

- **Undirected Graph**

  A graph whose edges do not have any direction is termed as undirected graph

  each edge ***ab*** = ***ba*** for undirected graph

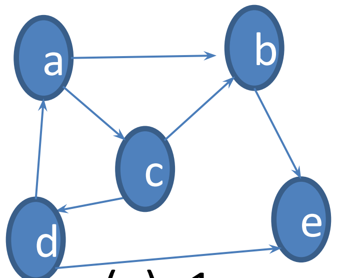
  edges are unordered pair.



- **Directed Graph**

  A digraph is a pair ***G=(V,E)*** where ***V*** is a finite set and ***E*** is an ordered pair of elements of ***V***

  each edge ab ≠ ba



- **Degree of Di-graph**

in-degree: number of edges incident on a vertex. in-deg(a)=1

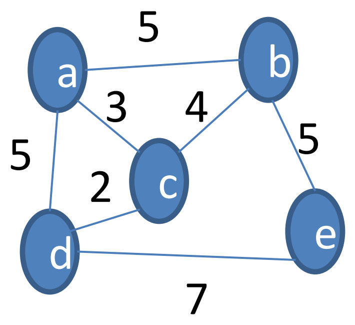
out-degree: number of edges emanating from a vertex. Out-deg(a)=2

Graph: Types

- **Weighted graph**

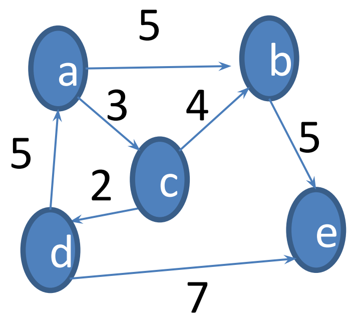
  Weighted graph is a pair ***(V,w)*** where ***V*** is a finite set of vertices and ***w*** is a function that assigns to each pair ***(x,y)*** of vertices either a positive integer or infinity. ***w*** is called the weight function, its value ***w(x,y)*** is interpreted as ***cost*** for moving directly from ***x*** to ***y***.

  The value w(x,y) =∞ indicates that there is no edge from ***x*** to ***y***



w(a,b)=w(b,a)=5
w(c,e)=∞

Weighted undirected graph



w(a,b)=5
w(b,a)=∞
w(c,e)=∞

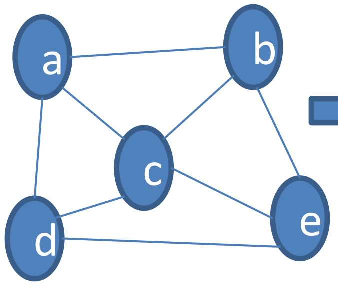
Weighted directed graph

Graph: Memory Representation

- **Adjacency Matrix**

The adjacency matrix of a graph G=(V,E) consists of a |V|X|V| matrix A=(a_{ij}) such that

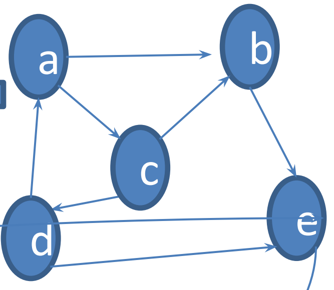
$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$



Undirected graph

	a	b	c	d	e
a	0	1	1	1	0
b	1	0	1	0	1
c	1	1	0	1	1
d	1	1	1	0	1
e	0	1	1	1	0

	a	b	c	d	e
a	0	1	1	0	0
b	0	0	0	0	1
c	0	1	0	1	0
d	1	0	0	0	1
e	0	0	0	0	1

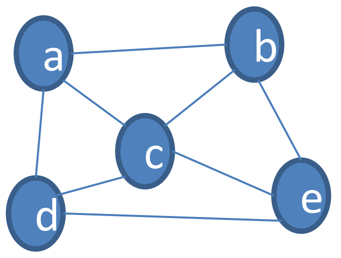


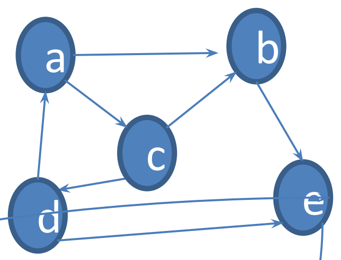
Directed graph

Graph: Memory Representation

- **Adjacency List**

  The adjacency list representation of a graph ***G=(V,E)*** consists of an array ***Adj*** of ***|V|*** lists, one for each vertex in ***V***. For each ***uϵ V***, the adjacency list ***Adj[u]*** contains all the vertices ***v*** such that there is an edge ***(u,v)ϵE***. That s ***Adj[u]*** consists of all the vertices adjacent to ***u*** in ***G***





$a \longrightarrow b \longrightarrow c \longrightarrow d$

$b \longrightarrow a \longrightarrow c \longrightarrow e$

$c \longrightarrow a \longrightarrow b \longrightarrow d \longrightarrow e$

$d \longrightarrow a \longrightarrow c \longrightarrow e$

$e \longrightarrow b \longrightarrow c \longrightarrow d$

$a \longrightarrow b \longrightarrow c$

$b \longrightarrow e$

$c \longrightarrow b \longrightarrow d$

$d \longrightarrow a \longrightarrow e$

$e \longrightarrow e$

Graph Traversal

- **Graph traversal**

  Traversing all the vertices of a graph starting from an arbitrary vertex

  It produces the set of all vertices that are **reachable from the starting node** of the graph (BFS)

  It produces the set of **all connected vertices** that belongs to the graph (DFS)

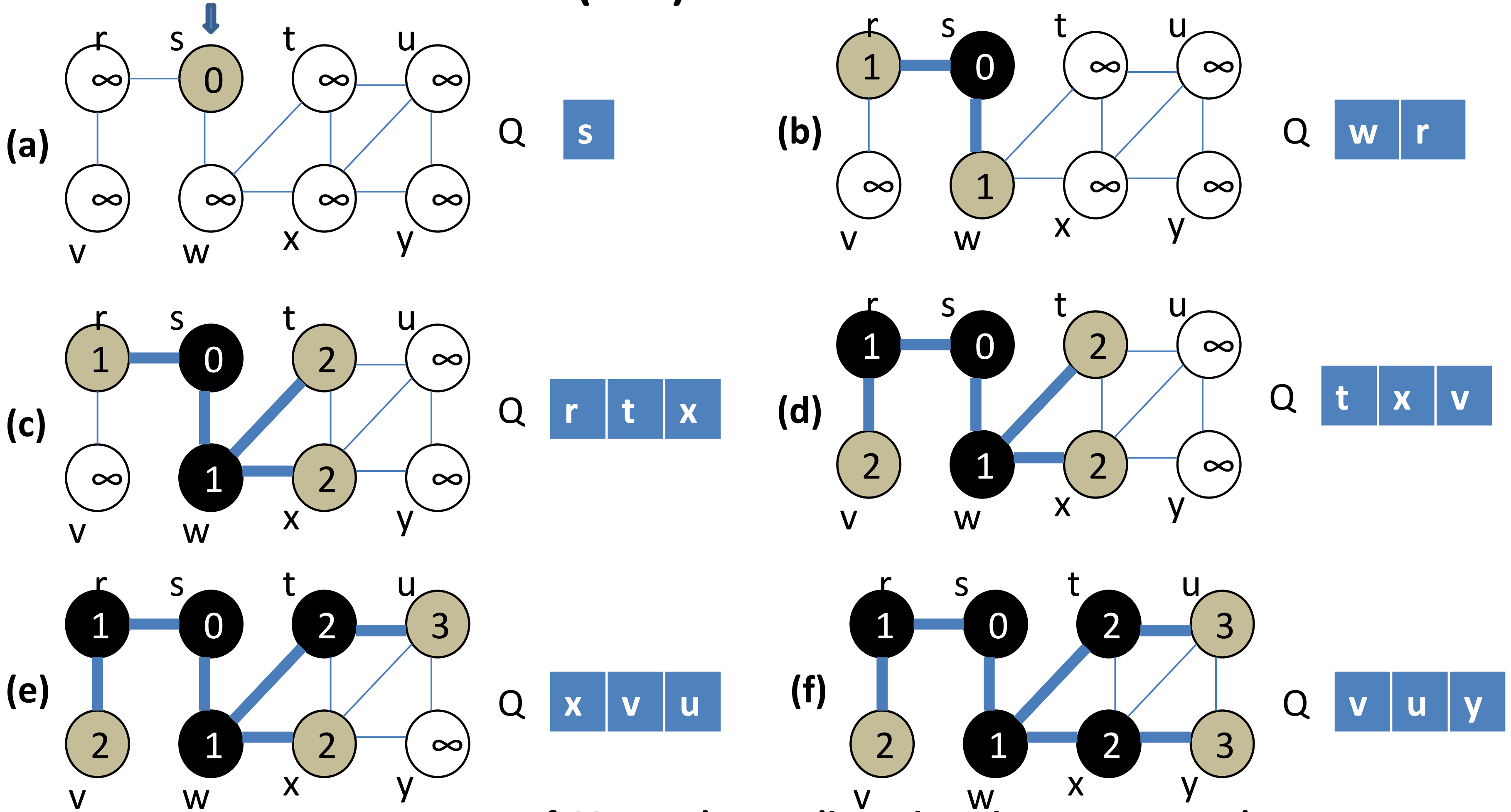
- **Traversal algorithm**
  - Breadth First Search (BFS)
  - Depth First Search (DFS)

Graph Traversal

- **Breadth First Search**
  - It starts from a source vertex ***s*** and systematically explore the graph ***G*** to discover every vertex that is reachable from ***s***
  - The algorithm discovers all vertices at distance ***k*** from ***s*** before discovering any vertices at distance ***k+1***. It searches the vertices that are across the breadth of the current discovered vertex
  - To keep track of the progress, BFS algorithm colors each vertex
    - White color node: still undiscovered
    - Gray node: discovered for the first time but still not processed
    - Black node: processed and written in O/P
  - Queue data structure is used to keep track of the discovered nodes (gray in color)
  - Initially, all nodes takes the color white. On completion of the algorithm, all the nodes will be black
  - BFS can be applied on both directed and undirected graph

Graph Traversal

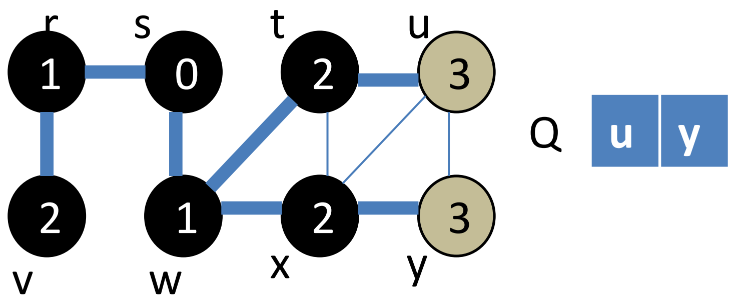
- **Breadth First Search (BFS)** mechanism

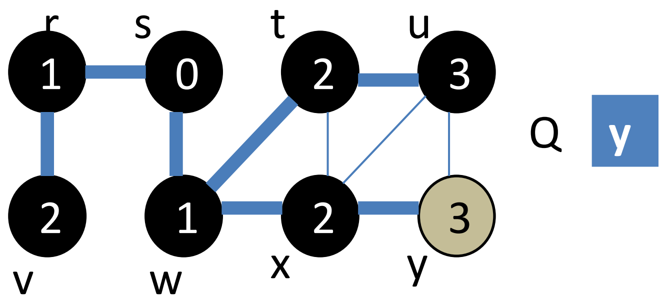


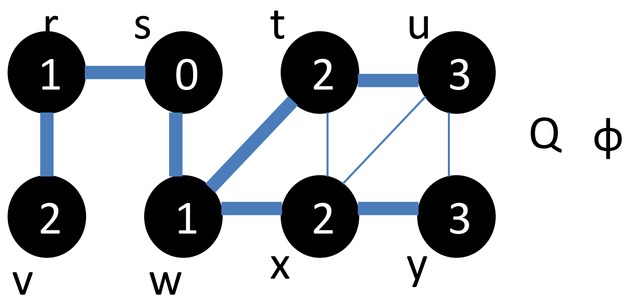
TECHNO INDIA UNIVERSITY
WESTBENGAL

Graph Traversal

- **Breadth First Search (BFS)** mechanism

(g)


(h)


(i)


BFS traversal:
s->w->r->t->x->v->u->y

O/P sequence may be different, it depends on the sequence in which I am considering the nodes of a particular breadth

Graph Traversal

- **Breadth First Search (BFS)** Algorithm

```
BFS(G,s)
 for each vertex u ϵ G.V – {s}
     u.color = WHITE
     u.d = ∞                // value assigned to node u
     u.π = NIL            // predecessor of node u
 s.color = GRAY
 s.d = 0
 s.π = NIL
 Q = ϕ                   // Queue is used to maintain the discovering nodes
 ENQUEUE(Q,s)   // discovered nodes are inserted into the Queue
 while  Q ≠ ϕ
      u= DEQUEUE(Q)   // delete the node to be processed
      for each v ϵ G.Adj[u]   // discovering all the adjacent nodes of the processing node
           if v.color == WHITE
               v.color = GRAY
               v.d = u.d + 1
               v.π = u
               ENQUEUE(Q,v)
      u.color = BLACK
```

TECHNO INDIA UNIVERSITY
WESTBENGAL

Graph Traversal

- **Breadth First Search (BFS)** Complexity analysis

1st for loop : The initialization process requires **O(V)** time as all the vertices are initialized each of which takes **O(1)** time

The operations of enqueuing and dequeuing takes **O(1)** time for a single vertex, total time taken **O(V)**

At the time of each dequeuing the adjacent nodes(or vertices) of the processing node are scanned through the connecting edges. Each edge is scanned at most once during the entire process. Hence, total edge scanning will take **O(E)**

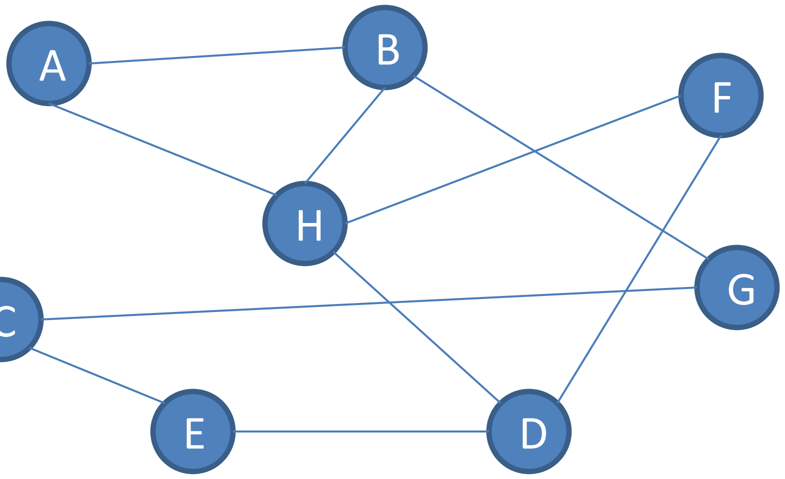
Thus, the total time complexity of BFS algorithm = **O(V+E)**

Queries?

TECHNO INDIA UNIVERSITY
WESTBENGAL

Practice Problem

1. Represent the following graph using
   a) Adjacency Matrix
   b) Adjacency List



2. Find the BFS traversal for the graph shown above. Consider node ***F*** as the starting node.