BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Data Structures and Algorithms
CS F211

Vishal Gupta
Department of Computer Science and Information Systems
Birla Institute of Technology and Science
Pilani Campus, Pilani

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

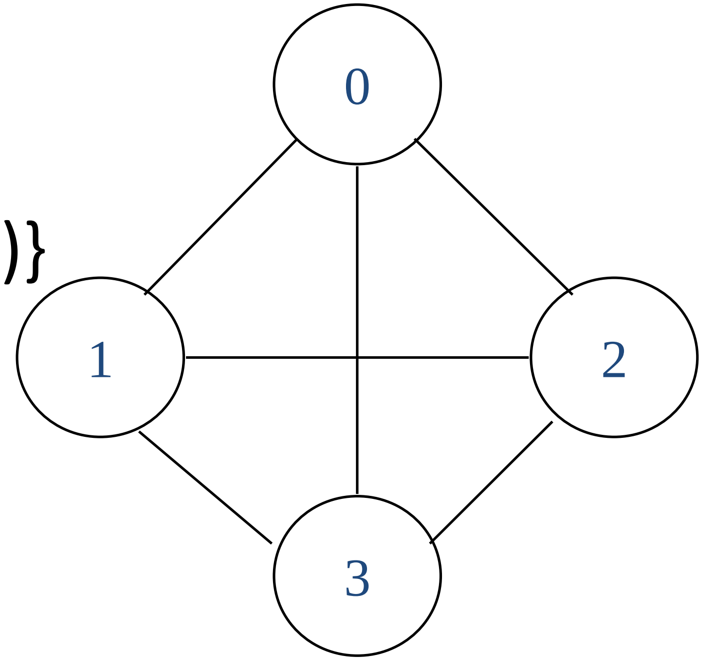
Agenda: Elementary Graph Algorithms

Definitions

- A graph G=(V,E), V and E are two sets
  - V: finite non-empty set of vertices
  - E: set of pairs of vertices, edges
- Undirected graph
  - The pair of vertices representing any edge is unordered. Thus, the pairs (u,v) and (v,u) represent the same edge
- Directed graph
  - each edge is represented by a ordered pairs <u,v>

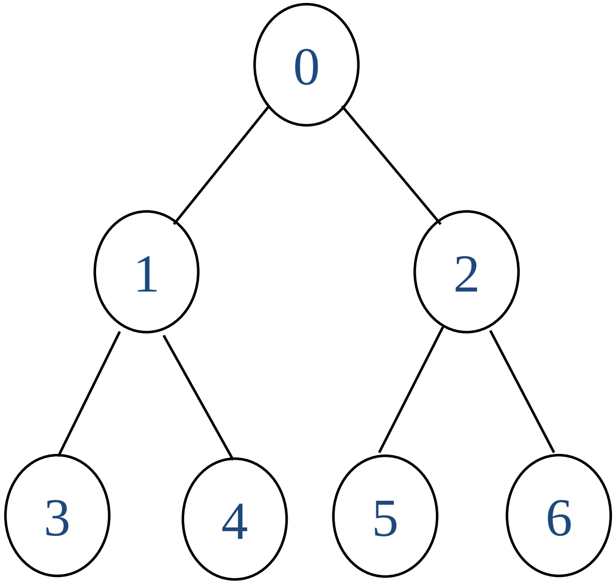
Examples of Graph G_1

- G_1
  - $V(G1)=\{0,1,2,3\}$
  - $E(G1)=\{(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)\}$



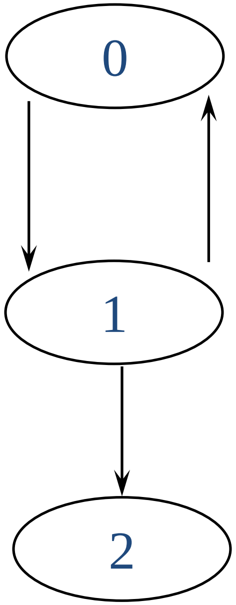
Examples of Graph G_2

- G_2
  - V(G2)={0,1,2,3,4,5,6}
  - E(G2)={(0,1),(0,2), (1,3),(1,4),(2,5),(2,6)}
- G_2 is also a tree
  - Tree is a special case of graph



Examples of Graph G_3

- G_3
  - V(G3)={0,1,2}
  - E(G3)={<0,1>,<1,0>,<1,2>}
- Directed graph (digraph)

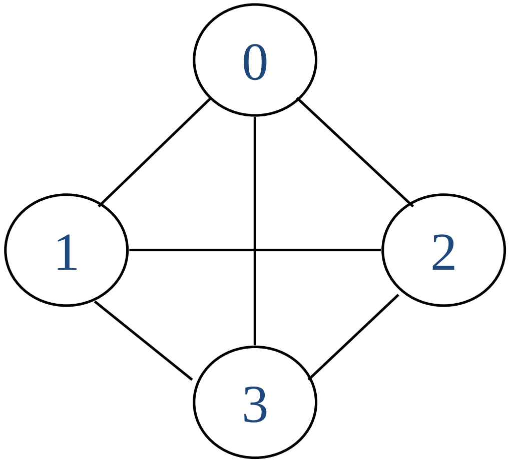


Adjacent and Incident

- If (u,v) is an edge in an undirected graph,
  - Adjacent: u and v are adjacent
  - Incident: The edge (u,v) is incident on vertices u and v
- If <u,v> is an edge in a directed graph
  - Adjacent: u is adjacent to v, and vu is adjacent from v
  - Incident: The edge <u,v> is incident on u and v

Cycle

- Cycle
  - a simple path, first and last vertices are same.
- 0,1,2,0 is a cycle
- Acyclic graph
  - No cycle is in graph



Degree

- Degree of Vertex
  - is the number of edges incident to that vertex
- Degree in directed graph
  - Indegree
  - Outdegree

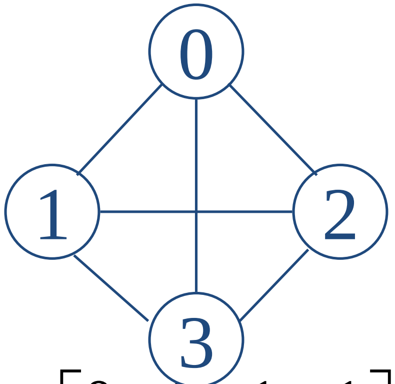
Graph Representations

- Adjacency Matrix
- Adjacency Lists

Adjacency Matrix

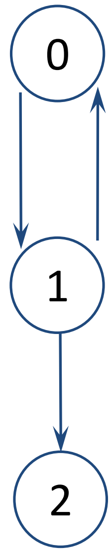
- Adjacency Matrix : let G = (V, E) with n vertices, $n \geq 1$. The adjacency matrix of G is a 2-dimensional $n \times n$ matrix, A
  - A(i, j) = 1 iff $(v_i, v_j) \in E(G)$ ($\langle v_i, v_j \rangle$ for a diagraph)
  - A(i, j) = 0 otherwise.
- The adjacency matrix for an undirected graph is symmetric; the adjacency matrix for a digraph need not be symmetric

Example



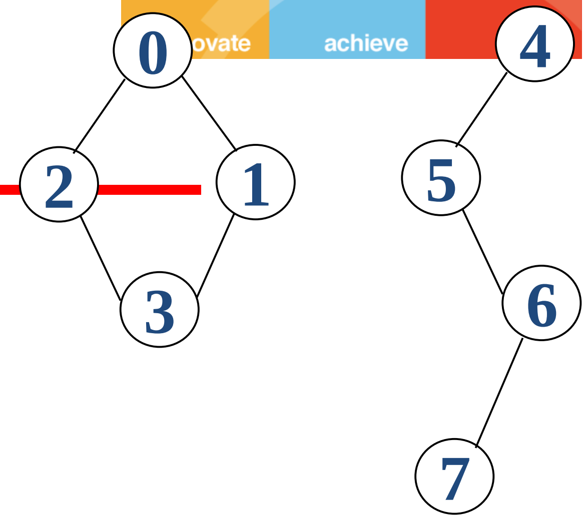
$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

G_1



$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

G2



$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

G_4

symmetric

undirected: $n^2/2$
directed: n^2

Merits of Adjacency Matrix

- From the adjacency matrix, to determine the connection of vertices is easy
- The degree of a vertex is $\sum_{j=0}^{n-1} adj\_mat[i][j]$
- For a digraph, the row sum is the out_degree, while the column sum is the in_degree

$$ind(vi) = \sum_{j=0}^{n-1} A[j,i] \quad outd(vi) = \sum_{j=0}^{n-1} A[i,j]$$

Adjacency Lists

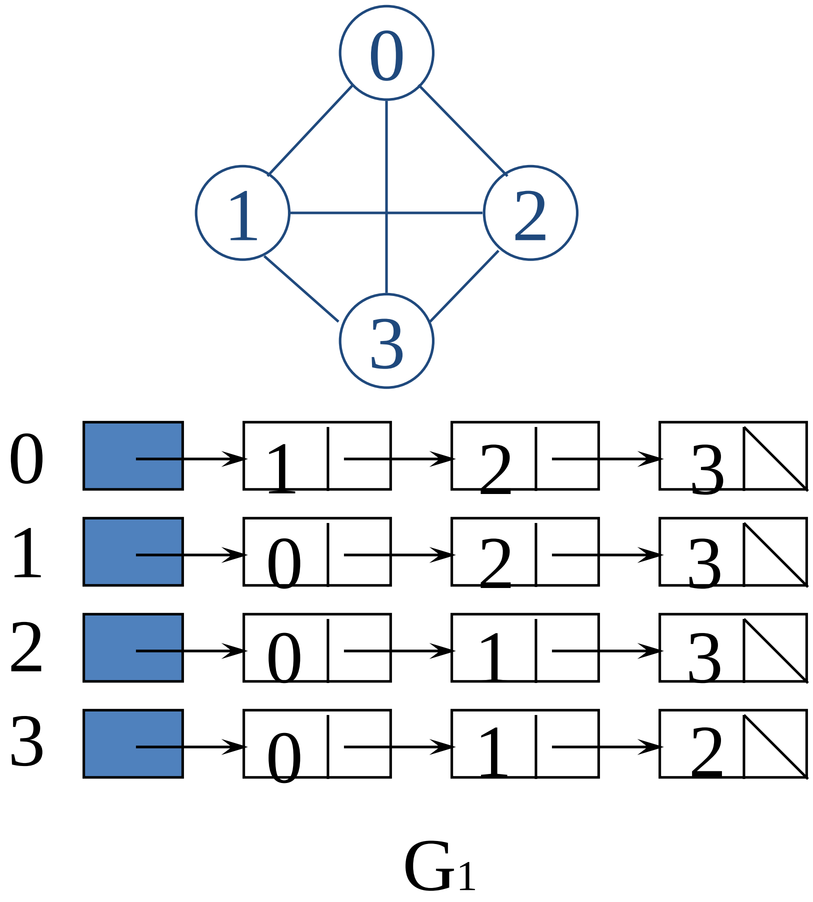
- Replace n rows of the adjacency matrix with n linked list

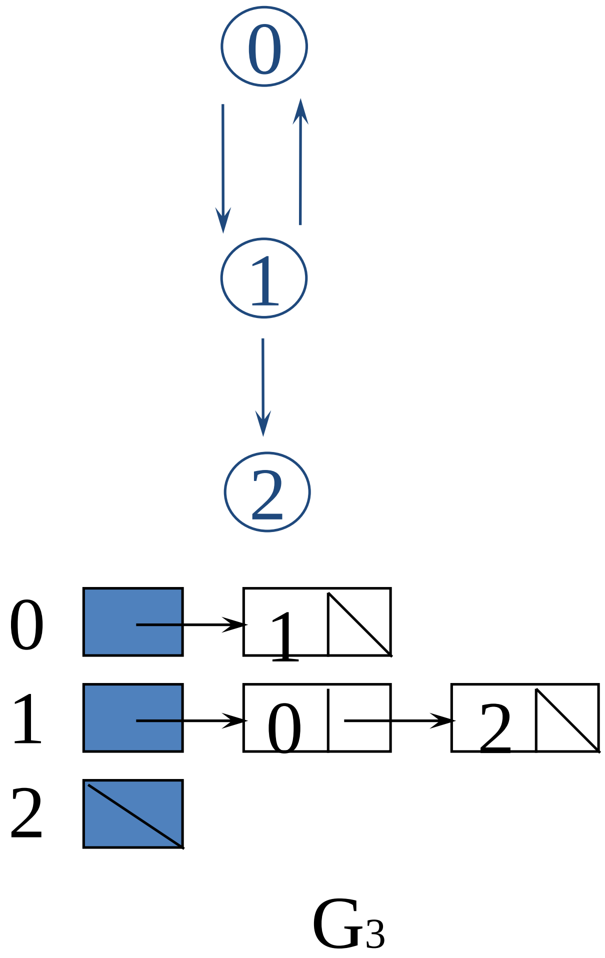
Data Structures for Adjacency Lists

Each row in adjacency matrix is represented as an adjacency list.

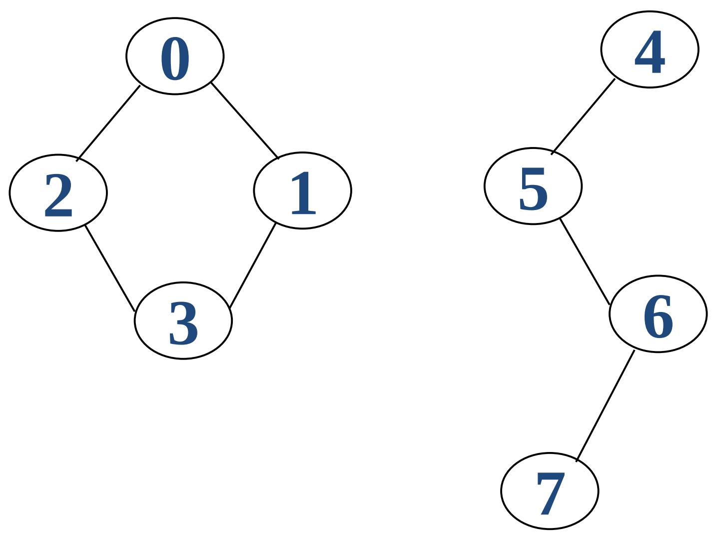
```
#define MAX_VERTICES 50
typedef struct node *node_pointer;
typedef struct node {
    int vertex;
    struct node *link;
};
node_pointer graph[MAX_VERTICES];
int n=0; /* vertices currently in use */
```

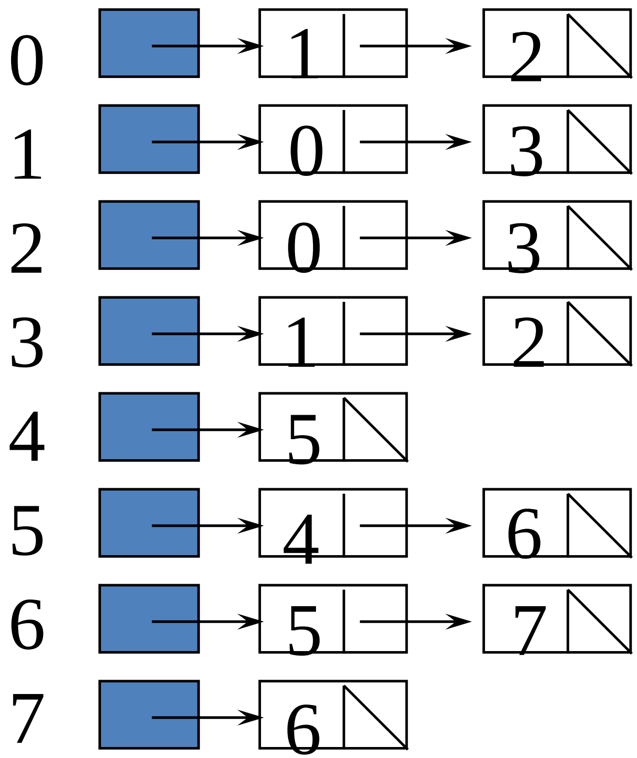
Example(1)





Example(2)

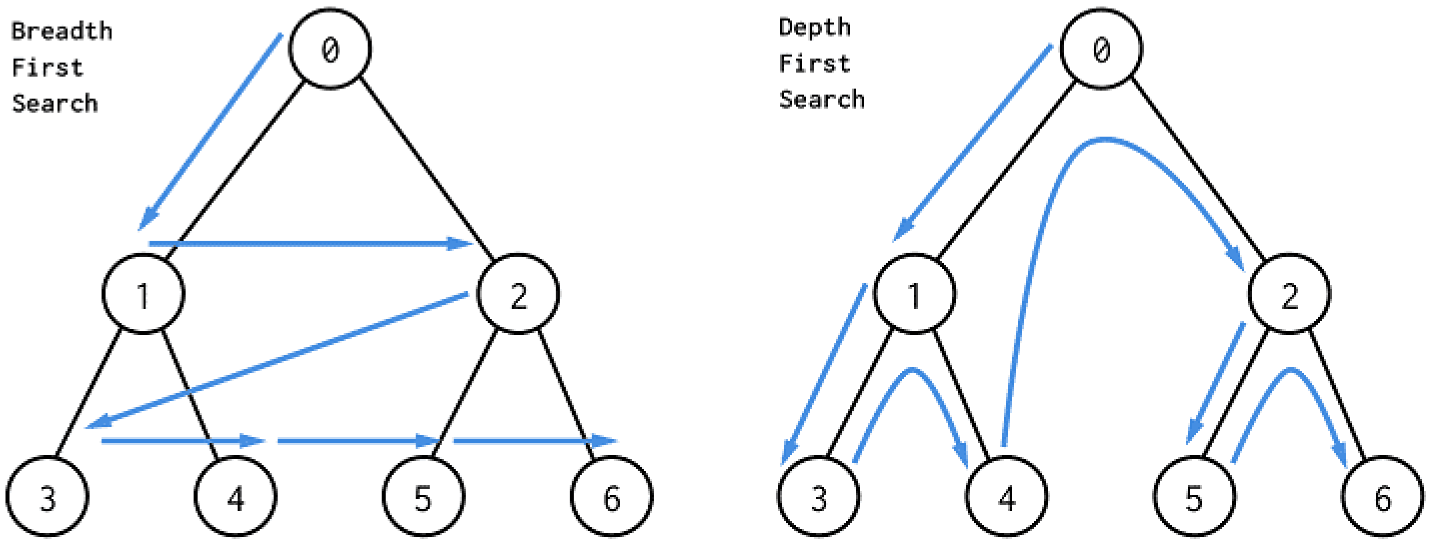




G_4

Interesting Operations

- degree of a vertex in an undirected graph
  - # of nodes in adjacency list
- # of edges in a graph
  - determined in O(n+e)
- out-degree of a vertex in a directed graph
  - # of nodes in its adjacency list
- in-degree of a vertex in a directed graph
  - traverse the whole data structure


Breadth
First
Search
0
1
2
3
4
5
6
Depth
First
Search
0
1
2
3
4
5
6

Breadth-first search

Breadth First Search

- ***Breadth-first search*** is one of the simplest algorithms for searching a graph.
- Dijkstra's single-source shortest-paths algorithm Prim's minimum-spanning-tree algorithm use ideas similar to those in BFS.

Breadth First Search

- Given a graph $G = (V, E)$ and a distinguished ***source*** vertex s, BFS systematically explores the edges of G to "discover" every vertex that is reachable from s.
- It computes the distance (fewest number of edges) from s to all such reachable vertices.
- It also produces a "breadth-first tree" with root s that contains all such reachable vertices.
- BFS discovers all vertices at distance k from s before discovering any vertices at distance $k + 1$.

Breadth First Search

```
Breadth_First_Search(G, s)
    for each vertex u ∈ G.V – {s}
        u.color = WHITE
        u.d = ∞
        u.π = NIL
    s.color = GRAY
    s.d = 0
    s.π = NIL
    Q = ϕ
    ENQUEUE (Q, s)
```

```
While (Q ≠ ϕ)
{
    u = DEQUEUE (Q)
    for each v ∈ G.Adj[u]
    {
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE (Q, v)
    }
    u.color = BLACK
}
```

Depth first search

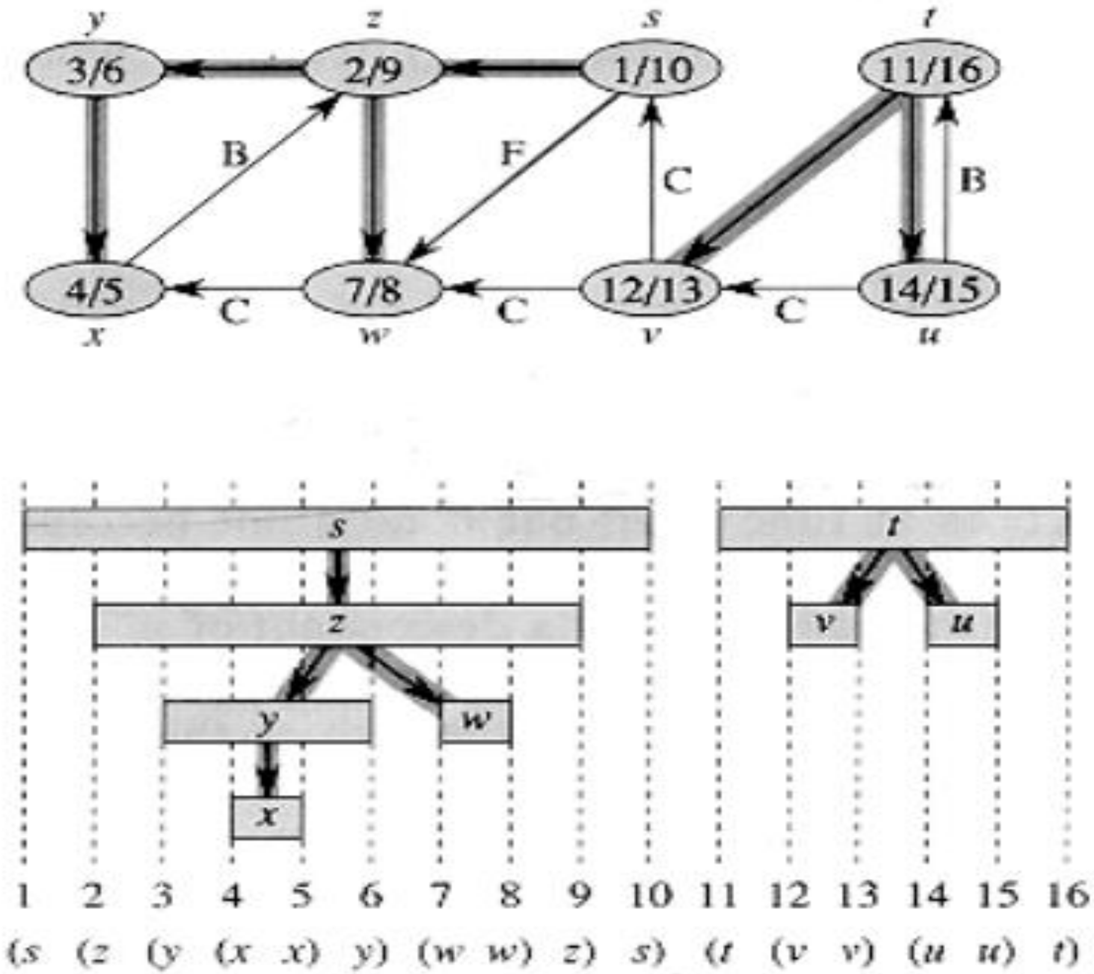
Global Variable: time

```
DEPTH_FIRST_SEARCH
   for each vertex u ∈ G.V
   {
       u.color = WHITE
       u.π =NIL
   }
   time = 0
   for each vertex u ∈ G.V
   {
        if u.color == WHITE
        DFS-VISIT (G, u)
   }
```

```
DFS-VISIT (G, u)
    time = time+1
    u.d = time
    u.color = GRAY
    for each v ∈ G.Adj[u]
    {
        if v.color == WHITE
            v.π = u
            DFS-VISIT (G, v)
    }
    u.color = BLACK
    time = time+1
    u.f = time
```

DFS: Properties

- discovery and finishing times have ***parenthesis structure***.



Parenthesis Theorem

In any depth first search of G = (V, E), for any two vertices u and v, exactly one of the following three conditions holds:

a) The intervals [u.d, u.f] and [v.d, v.f] are entirely disjoint, and neither u nor v is a descendant of the other in the depth first forest.
b) The interval [u.d, u.f] is contained entirely within the interval [v.d, v.f], and u is a descendent of v in a depth first tree, or
c) The interval [v.d, v.f] is contained entirely within the interval [u.d, u.f], and v is a descendent of u in a depth first tree

Corollary: Vertex v is a proper descendant of vertex u in the depth first forest for a (directed or undirected) graph G if and only if

$$u.d < v.d < v.f < u.f$$

Classification of Edges

- **DFS can be used to classify the edges of G into:**

a) **Tree Edge**: Edge (u, v) is a tree edge if v was first discovered by exploring edge (u, v).

b) **Back edges:** Edges (u, v) connecting a vertex u to an ancestor v in a depth-first tree. Self-loops, which may occur in directed graphs, are considered to be back edges.

c) **Forward edges:** Those non-tree edges (u, v) connecting a vertex u to a descendant v in a depth-first tree.

d) **Cross edges** are all other edges. They can go between vertices in the same depth-first tree, as long as one vertex is not an ancestor of the other, or they can go between vertices in different depth-first trees.

- **Edge (u, v) can be classified by the color of the vertex v that is reached when the edge is first explored**

1. **WHITE** indicates a tree edge,
2. **GRAY** indicates a back edge, and
3. **BLACK** indicates a forward or cross edge.

Properties

- In a depth-first search of an undirected graph G, every edge of G is either a tree edge or a back edge.
- A directed graph is acyclic if and only if a depth-first search yields no “back” edges.
- For a weighted graph, DFS traversal of the graph produces the minimum spanning tree and all pair shortest path tree.
- We can specialize the DFS algorithm to find a path between two given vertices u and z.
    i. Call DFS(G, u) with u as the start vertex.
    ii. Use a stack S to keep track of the path between the start vertex and the current vertex.
    iii. As soon as destination vertex z is encountered, return the path as the contents of the stack

Properties

- In a depth-first search of an undirected graph G, every edge of G is either a tree edge or a back edge.
- A directed graph is acyclic if and only if a depth-first search yields no “back” edges.
- For a weighted graph, DFS traversal of the graph produces the minimum spanning tree and all pair shortest path tree.
- We can specialize the DFS algorithm to find a path between two given vertices u and z.
    1. Call DFS(G, u) with u as the start vertex.
    2. Use a stack S to keep track of the path between the start vertex and the current vertex.
    3. As soon as destination vertex z is encountered, return the path as the contents of the stack

Topological sort

- A topological sort of a DAG G = (V, E) is a linear ordering of all its vertices such that if G contains an edge (u, v), then u appears before v in the ordering.
- A topological sort of a graph can be viewed as an ordering of its vertices along a horizontal line so that all directed edges go from left to right.
- If the graph is not acyclic, then no linear ordering is possible.

TOPOLOGICAL-SORT (G)

1. **call DFS(G) to compute finishing times u.f for each vertex u**
2. **As each vertex is finished, insert it onto the front of a linked list**
3. **Return the linked list of vertices**

Topological sort