

A large, abstract network graph occupies the left two-thirds of the slide. It consists of numerous small, semi-transparent circular nodes of varying sizes scattered across a dark purple-to-orange gradient background. A dense web of thin, light-colored lines connects these nodes, forming a complex web-like structure.

Foundations of Algorithm

SCS1308

Dr. Dinuni Fernando
PhD
Senior Lecturer



Graph Theory and Representation

Graph Algorithms

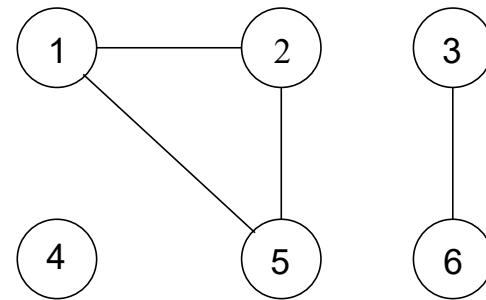
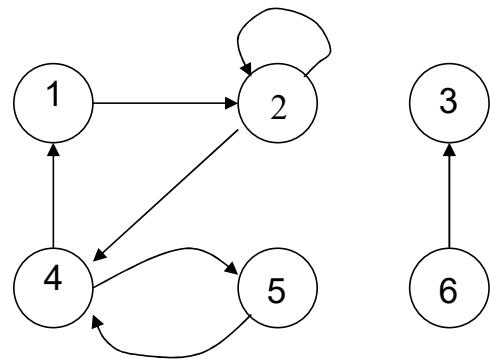
- Graphs and fundamental theorems about Graphs
- Graph implementation
- Graph Algorithms
 - Shortest paths
 - Minimum spanning tree

What can graphs model?

- Cost of wiring electronic components together.
- Shortest route between two cities.
- Finding the shortest distance between all pairs of cities in a road atlas.
- Flow of material: liquid flowing through pipes, current through electrical networks, information through communication networks, parts through an assembly line, etc.
- State of a machine.
- Used in Operating systems to model resource handling (deadlock problems).
- Used in compilers for parsing and optimizing the code.

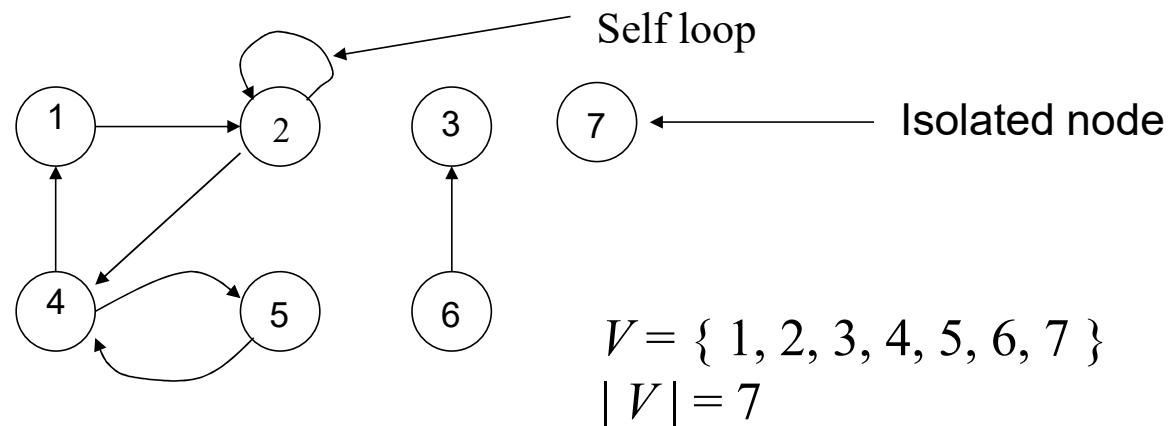
What is a Graph?

- Informally a *graph* is a set of nodes joined by a set of lines or arrows.



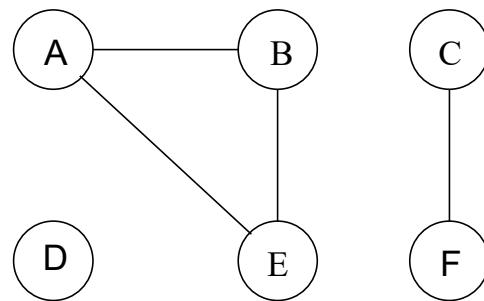
A **directed graph**, also called a **digraph G**, is a pair (V, E) where V is a finite set of vertices and E is a set of *directed edges*.

An edge from node a to node b is denoted by the **ordered** pair (a, b) .



$$E = \{ (1,2), (2,2), (2,4), (4,5), (4,1), (5,4), (6,3) \}$$
$$| E | = 7$$

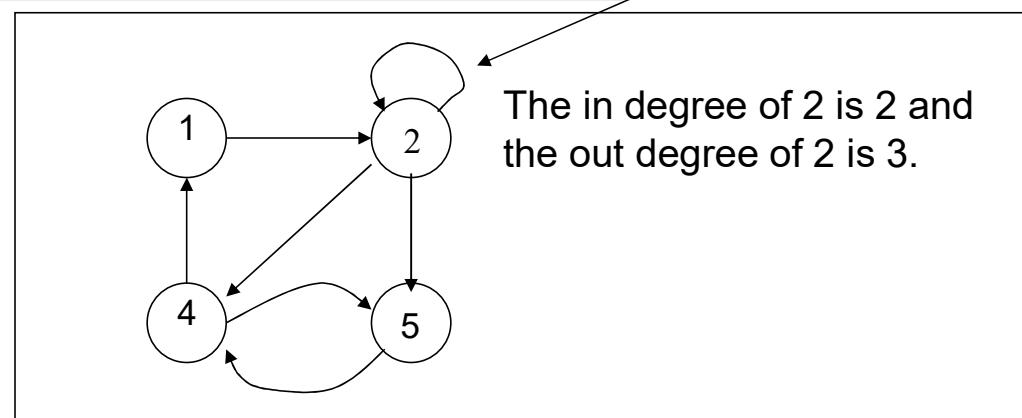
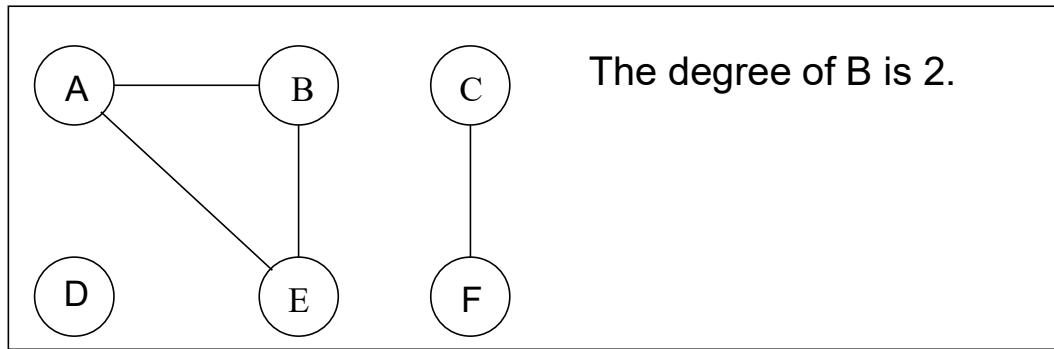
Undirected graph $G = (V, E)$: Unlike a digraph, E consists of undirected edges. So, edge $(A, B) = \text{edge } (B, A)$.



$$V = \{ A, B, C, D, E, F \}$$
$$|V| = 6$$

$$E = \{ \{A, B\}, \{A, E\}, \{B, E\}, \{C, F\} \}$$
$$|E| = 4$$

- The *degree* of a vertex in an undirected graph is the number of edges incident on it.
- In a directed graph, the *out degree* of a vertex is the number of edges leaving it and the *in degree* is the number of edges entering it.



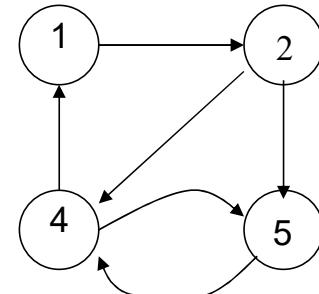
Cyclic and Acyclic

A path from a vertex to itself is called a cycle

(e.g., $v1 \rightarrow v2 \rightarrow v4 \rightarrow v1$)

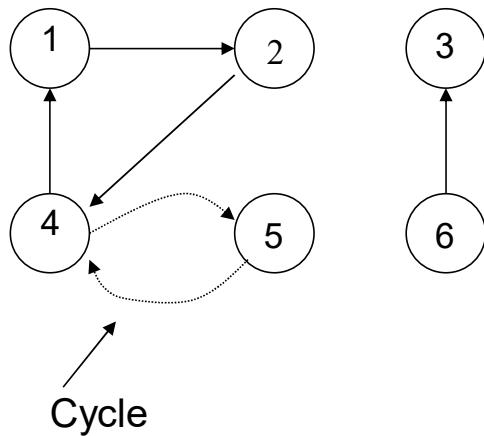
If a graph contains a cycle, it is cyclic

Otherwise, it is acyclic

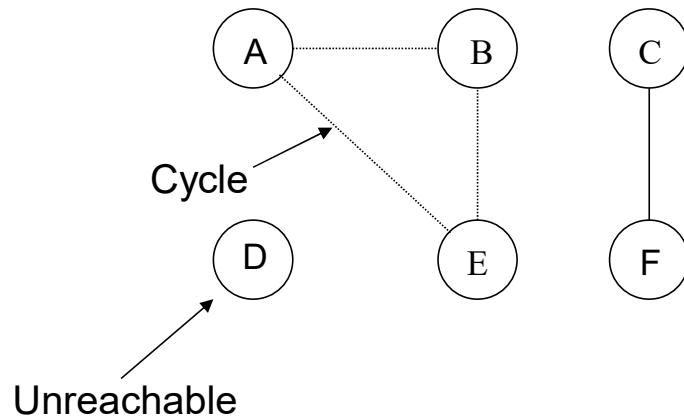


*A path is **simple** if it never passes through the same vertex twice.*

A **path** is a sequence of vertices such that there is an edge from each vertex to its successor. A path from a vertex to itself is called a **cycle**. A graph is called **cyclic** if it contains a cycle; otherwise it is called **acyclic**. A path is **simple** if each vertex is distinct.



Simple path from 1 to 5
 $= (1, 2, 4, 5)$
 or as in our text
 $((1, 2), (2, 4), (4, 5))$



If there is path p from u to v then we say v is **reachable** from u via p .

Simple Graphs

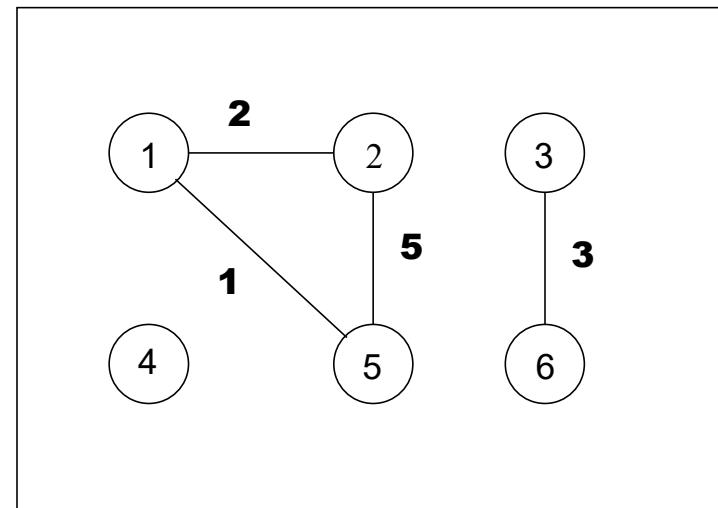
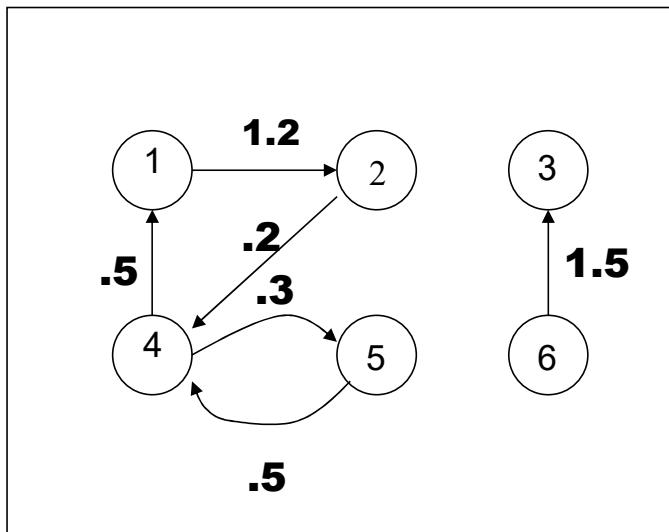
- *Simple graphs* are graphs without multiple edges or self-loops. We will consider only simple graphs.
- Proposition: If G is an undirected graph then

$$\sum_{v \in G} \deg(v) = 2 |E|$$

- Proposition: If G is a digraph then

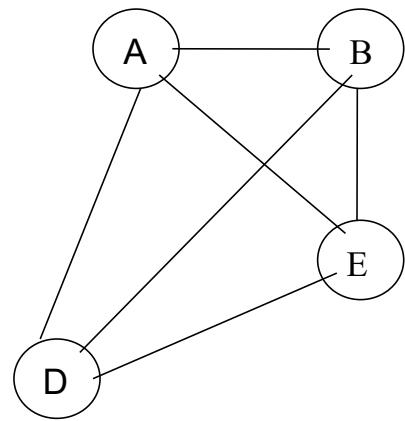
$$\sum_{v \in G} \text{indeg}(v) = \sum_{v \in G} \text{outdeg}(v) = |E|$$

A ***weighted graph*** is a graph for which each edge has an associated ***weight***, usually given by a ***weight function*** $w: E \rightarrow \mathbf{R}$.



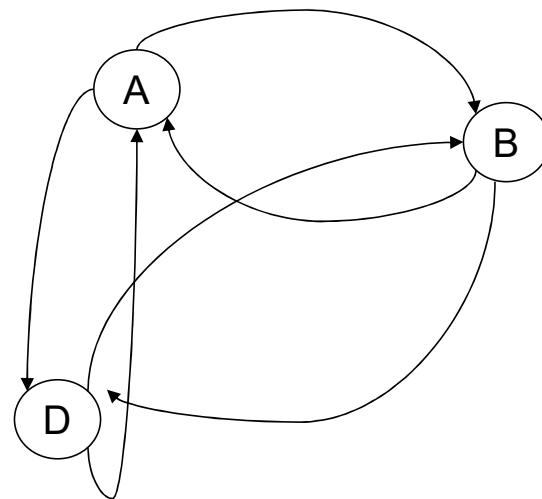
If (u, v) is an edge in a graph G , we say that vertex v is **adjacent** to vertex u .

A **complete graph** is an undirected/directed graph in which every pair of vertices is adjacent.



4 nodes and $(4*3)/2$ edges

V nodes and $V*(V-1)/2$ edges



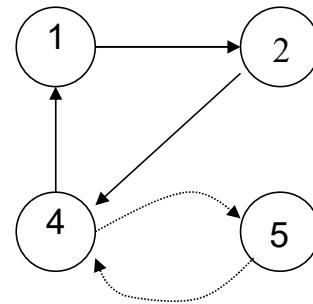
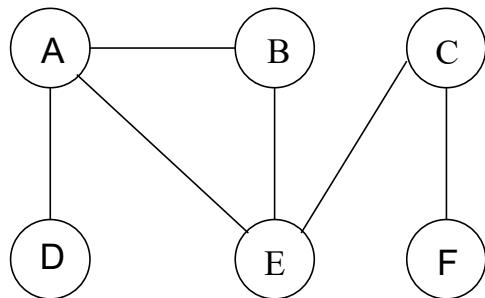
3 nodes and $3*2$ edges

V nodes and $V*(V-1)$ edges

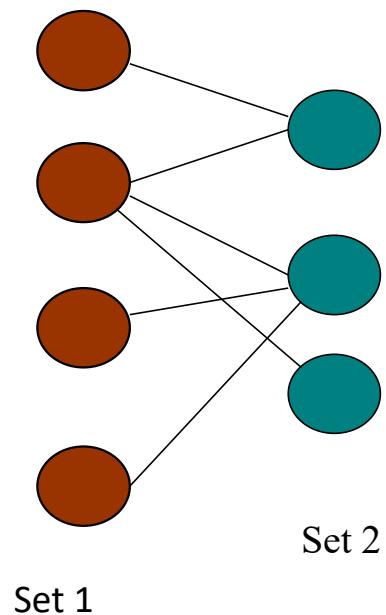
An undirected graph is ***connected*** if you can get from any node to any other by following a sequence of edges, i.e., a path.

A directed graph is ***strongly connected*** if there is a directed path from any node to any other node.

- A graph is ***sparse*** if $|E| \approx |V|$
- A graph is ***dense*** if $|E| \approx |V|^2$



A **bipartite graph** is an undirected graph $G = (V, E)$ in which V can be partitioned into 2 sets V_1 and V_2 such that $(u, v) \in E$ implies either $u \in V_1$ and $v \in V_2$ OR $v \in V_1$ and $u \in V_2$.

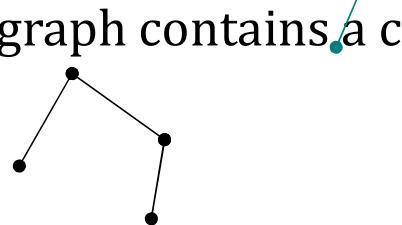
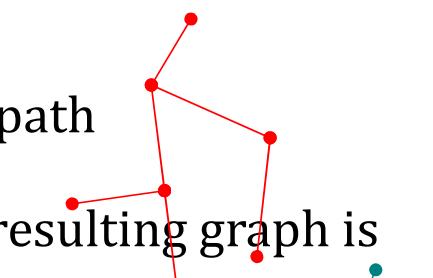


A **free tree** is an acyclic, connected, undirected graph. A **forest** is an acyclic undirected graph. A **rooted tree** is a tree with one distinguished node, **root**.

Let $G = (V, E)$ be an undirected, acyclic, connected graph.

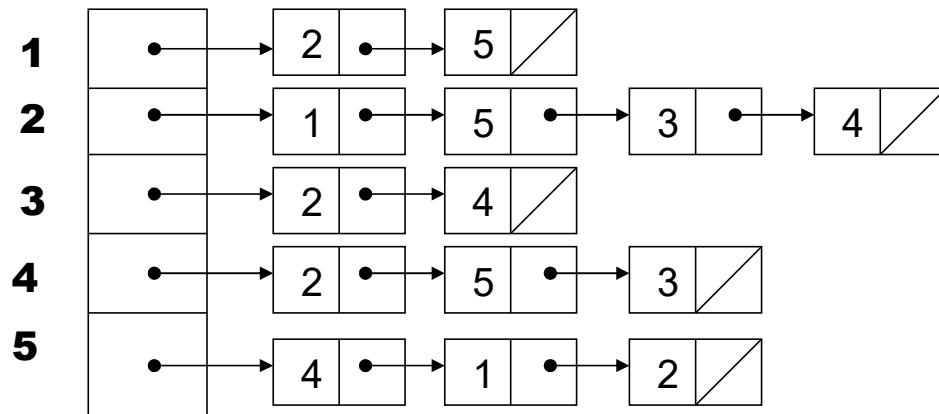
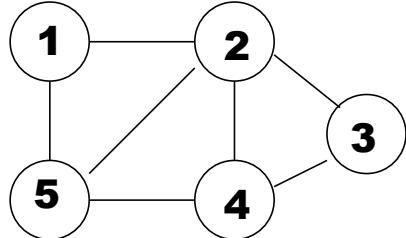
The following statements are equivalent.

- G is a tree
- Any two vertices in G are connected by unique simple path
- G is connected, acyclic, and $|E| = |V| - 1$
- G is connected, but if any edge is removed from E , the resulting graph is disconnected
- G is acyclic, but if any edge is added to E , the resulting graph contains a cycle.

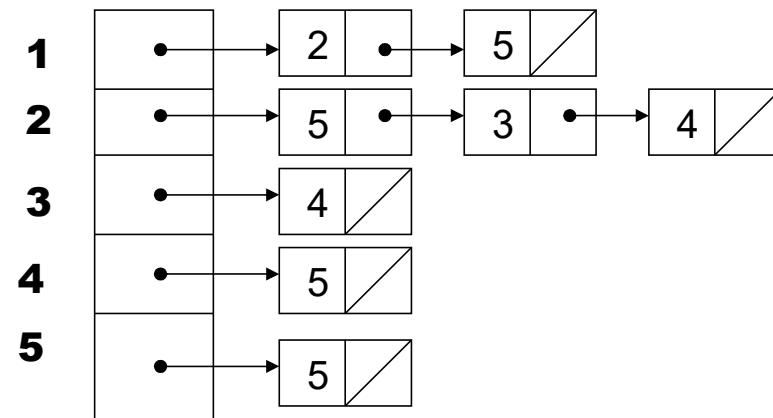
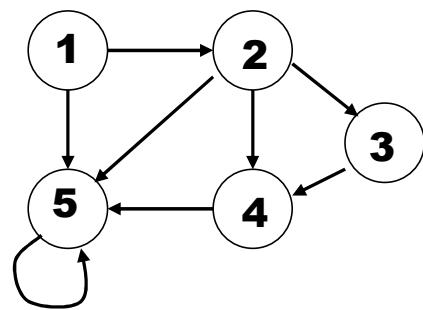


Implementation of a Graph.

- ***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 $u \in V$, $ADJ[u]$ points to all its adjacent vertices.



Adjacency-list representation for a directed graph.



Variation: Can keep a second list of edges coming into a vertex.

Adjacency lists

- Property
 - Saves space for sparse graphs. Most graphs are sparse.
 - “Visit” edges that start at v
 - Must traverse linked list of v
 - Size of linked list of v is $\text{degree}(v)$
 - Order: $\Theta(\text{degree}(v))$

Adjacency List

- Storage

- We need V pointers to linked lists
- For a directed graph the number of nodes (or edges) contained (referenced) in all the linked lists is

$$\sum_{v \in V} (\text{out-degree } (v)) = |E|.$$

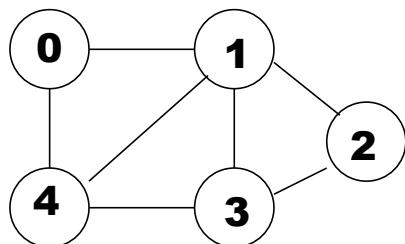
So we need $\Theta(V + E)$

- For an undirected graph the number of nodes is

$$\sum_{v \in V} (\text{degree } (v)) = 2 |E|$$

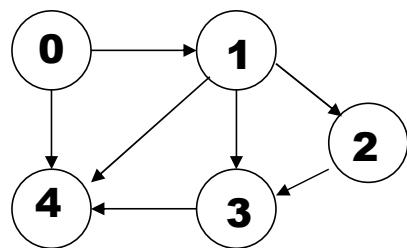
Also $\Theta(V + E)$

Adjacency-matrix representation of a graph $G = (V, E)$ is a $|V| \times |V|$ matrix $A = (a_{ij})$ such that $a_{ij} = 1$ if $(i, j) \in E$ and 0 otherwise.



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

Adjacency Matrix Representation for a Directed Graph



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

Adjacency Matrix Representation

- Advantage:
 - Saves space on pointers for **dense, un-weighted** graphs
 - Just one bit per matrix element
 - **Faster lookup**
 - Is there an edge (v, u) ? \Leftrightarrow adjacency [i] [j] == true?
 - So $\theta(1)$
- Disadvantage:
 - Waste space for **sparse, weighted** graphs
 - Size of the adjacency matrix is $|V|^2$
 - “Visit” all the edges that start at v
 - Row v of the matrix must be traversed.
 - So $\theta(|V|)$.

Adjacency Matrix Representation

- Storage
 - $\Theta(V^2)$
 - For undirected graphs you can only use $1/2(V^2)$ storage, since the adjacency matrix of an undirected graph is symmetric

Graph traversals

- Breadth first search
- Depth first search

Breadth first search

- Given a graph $G=(V,E)$ and a *source vertex* s , BFS explores the edges of G to visit each node of G reachable from s .
- Idea - Expand a *frontier* one step at a time.
- *Frontier* is a FIFO queue
 - $O(1)$ time to update

Breadth first search

- Computes the *shortest distance (dist)* from s to any reachable node
- Computes a *breadth first tree (of parents)* with root s that contains all the reachable vertices from s
- To get $O(|V|+|E|)$ if we use an adjacency list representation. If we used an adjacency matrix, it would be $O(|V|^2)$

Coloring the nodes

- We use colors (*white*, *gray* and *red*) to denote the state of the node during the search
- A node is ***white*** if it has not been reached (visited)
- *Visited* nodes are *gray* or *red*. ***Gray*** nodes are at the frontier of the search. ***Red*** nodes are fully explored nodes

BFS - initialize

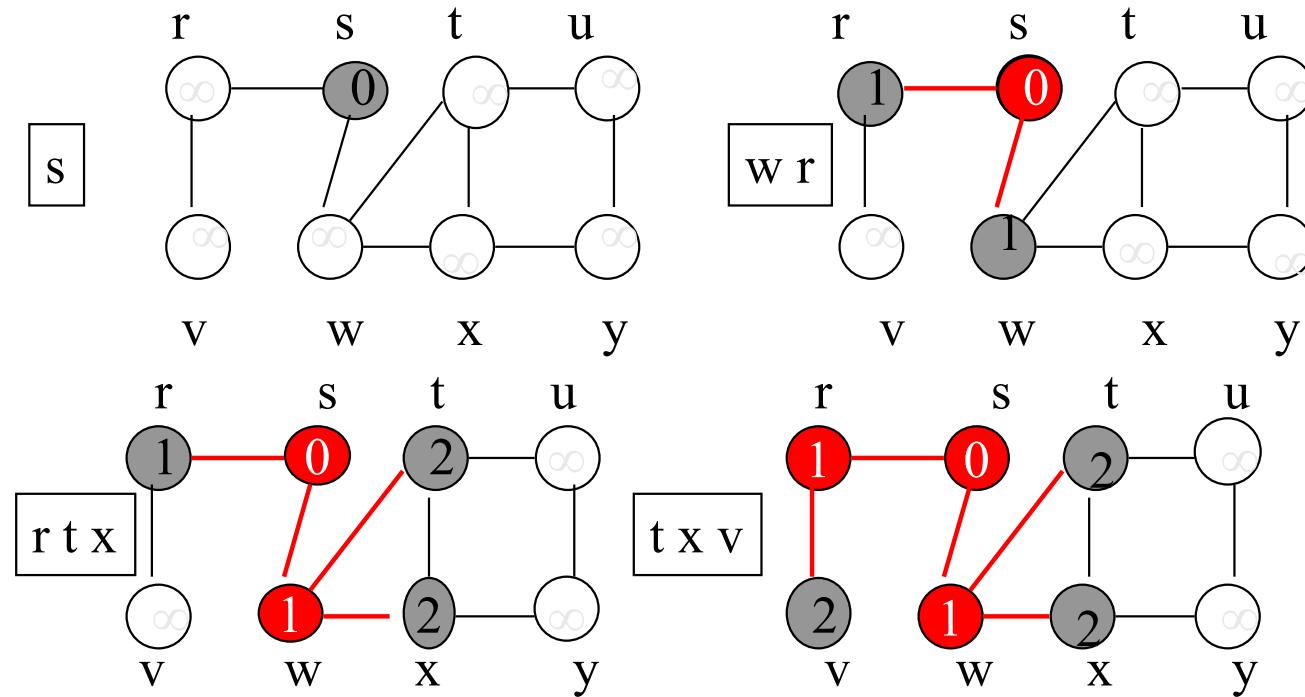
```
procedure BFS(G:graph; s:node; var color:c_array; dist:iarray;  
parent:parray);  
for each vertex u do  
    color[u]=white; dist[u]=∞;           Θ(V)  
    parent[u]=nil;  
end for  
color[s]=gray; dist[s]=0;  
init(Q); enqueue(Q, s);
```

BFS - main

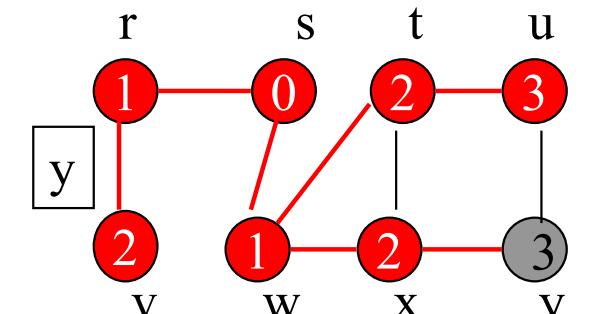
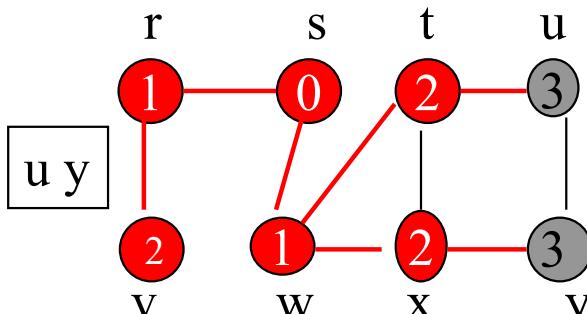
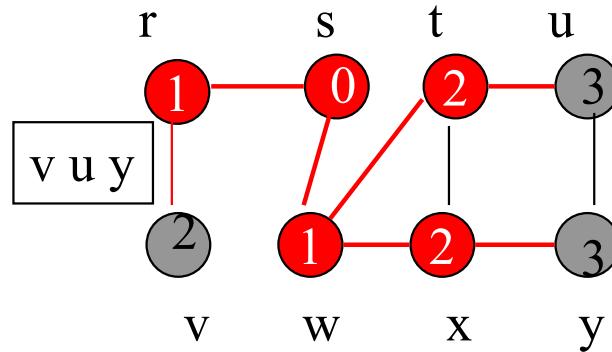
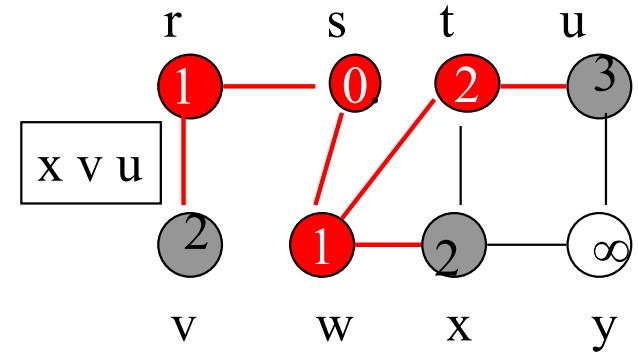
```
while not (empty(Q)) do
    u=head(Q);
    for each v in adj(u) do          O(E)
        if color[v]=white then
            color[v]=gray; dist[v]=dist[u]+1;
            parent[v]=u; enqueue(Q, v);
    dequeue(Q); color[u]=Red; print "u";
end BFS
```

$$\sum_{u \in V} \sum_{v \in ADJ[u]} 1 = \sum_{u \in V} |ADJ[u]| = \sum_{u \in V} \deg{u} = O(E)$$

BFS example



BFS example



now y is removed from the Q and colored red

Analysis of BFS

- Initialization is $\Theta(|V|)$.
- Each **node** can be **added** to the queue at most **once** (it needs to be white), and its **adjacency list** is **searched only once**. At most all adjacency lists are searched.
- If graph is undirected each edge is reached twice, so loop repeated at most $2|E|$ times.
- If a graph is directed each edge is reached exactly once. So the loop is repeated at most $|E|$ times.
- Worst case time $O(|V|+|E|)$