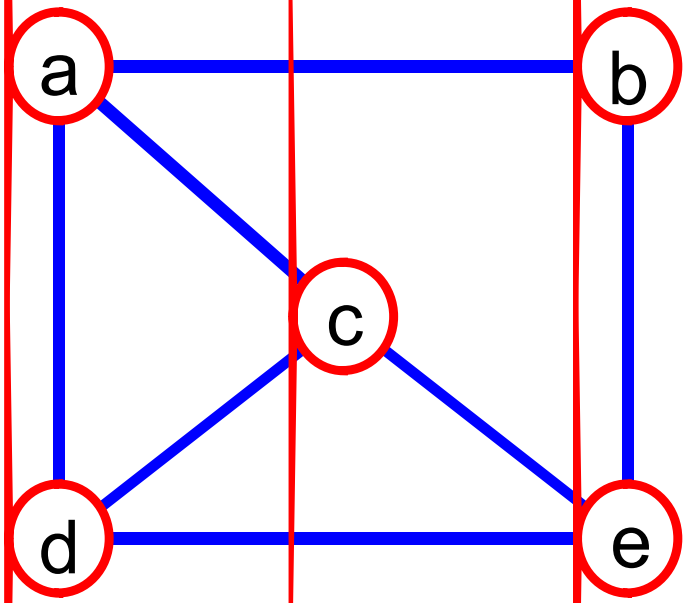
UNIT – V

GRAPHS

Graph Definition – Graph terminologies - Representation of Graphs – Graph traversal -Topological Sort – Shortest Path algorithms: Dijkstra's and Floyd's algorithms - Minimum Spanning Tree : Prim's and Kruskal's algorithms.

What is a Graph?

- A graph G = (V,E) is composed of:
  V: set of vertices
  E: set of edges connecting the vertices in V
- An edge e = (u,v) is a pair of vertices
- Example:

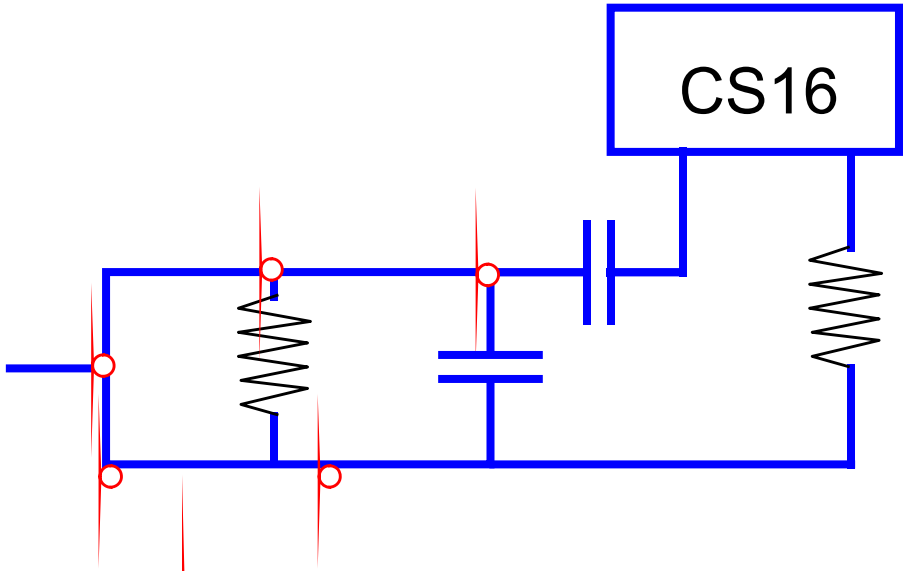


V= {a,b,c,d,e}

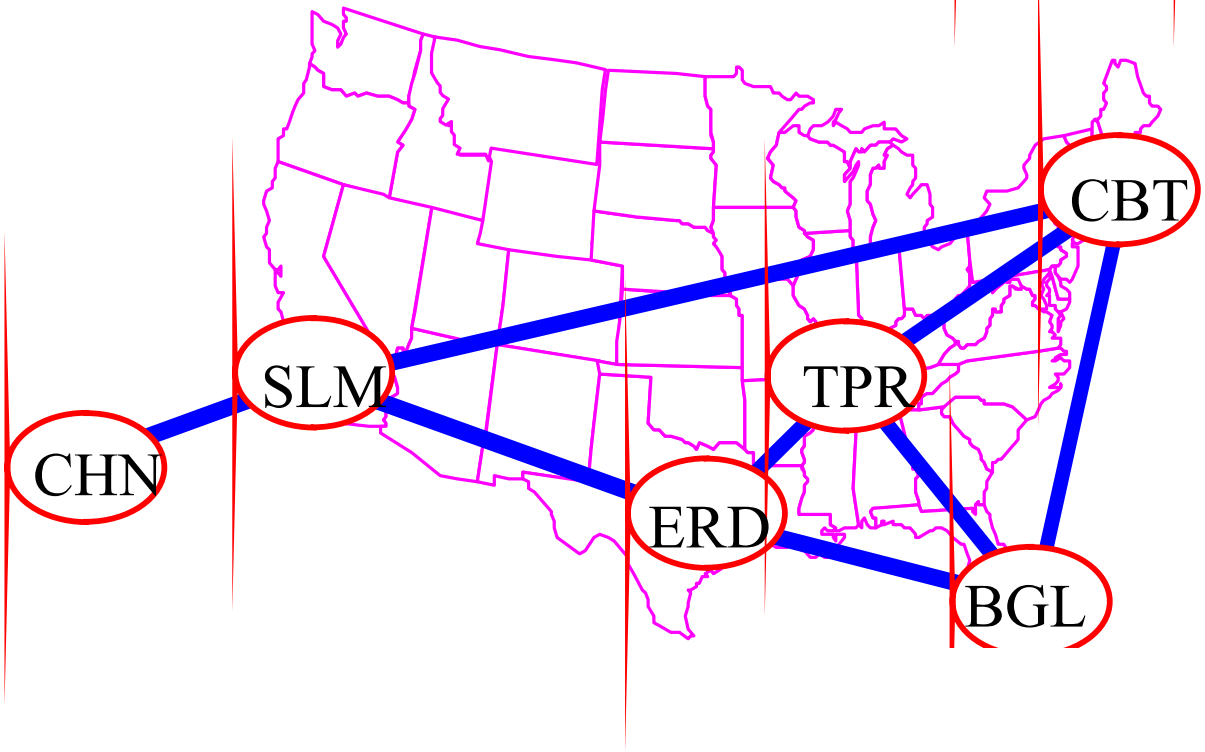
E= {(a,b),(a,c),(a,d),
(b,e),(c,d),(c,e),
(d,e)}

Applications

- electronic circuits



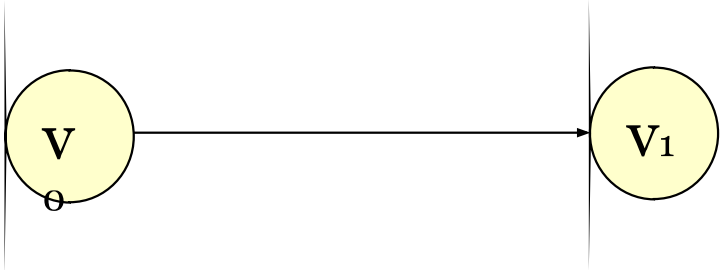
- networks (roads, flights, communications)



Terminology: Adjacent and Incident

- If (v_0, v_1) is an edge in an **undirected graph**,
  - v_0 and v_1 are adjacent
  - The edge (v_0, v_1) is incident on vertices v_0 and v_1

- If $\langle v_0, v_1 \rangle$ is an edge in a **directed graph**
  - v_0 is adjacent to v_1, and v_1 is adjacent from v_0
  - The edge $\langle v_0, v_1 \rangle$ is incident on v_0 and v_1



Directed vs. Undirected Graph

- An undirected graph is one in which the pair of vertices in a edge is unordered

$(v_0, v_1) = (v_1, v_0)$

- A directed graph is one in which each edge is a directed pair of vertices

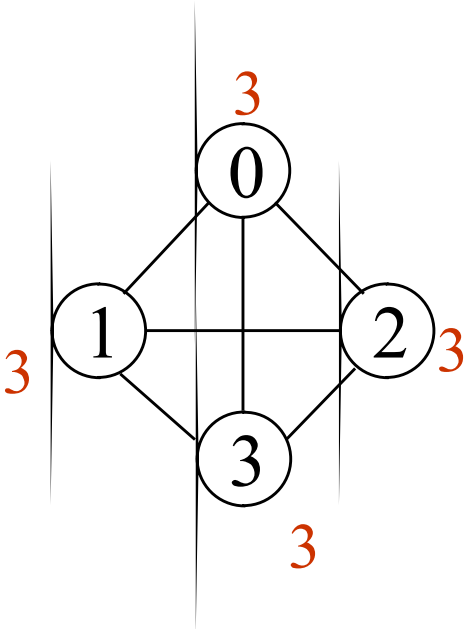
$(v_0, v_1) \neq (v_1, v_0)$

Terminology: Degree of a Vertex

- The degree of a vertex is the number of edges incident to that vertex
- For directed graph,
  - the in-degree of a vertex v is the number of edges that have v as the head
  - the out-degree of a vertex v is the number of edges that have v as the tail

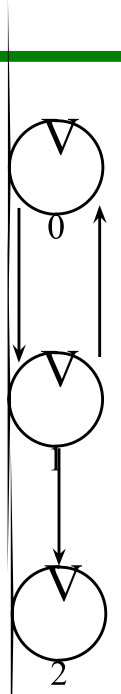
Examples

G_1



Undirected graph

G_2



in:1, out: 1

in: 1, out: 2

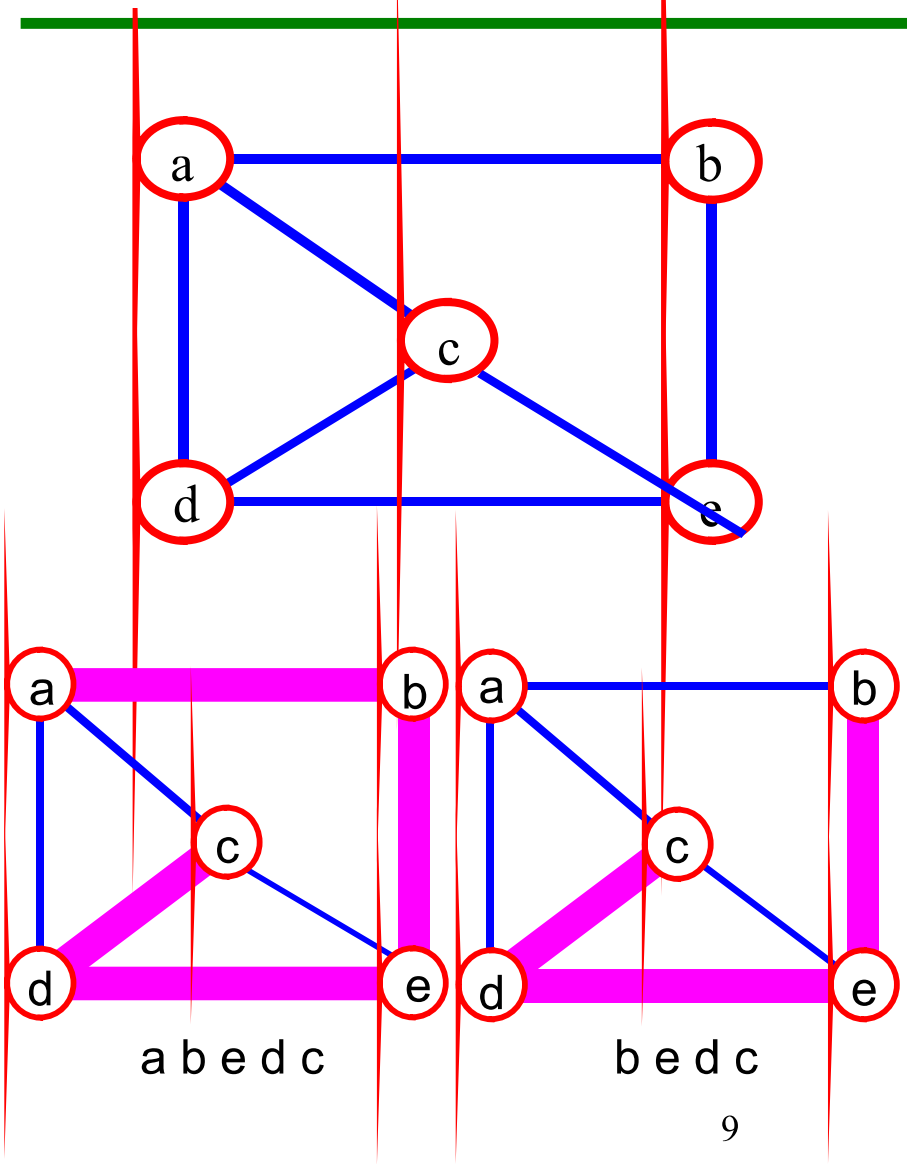
in: 1, out: 0

Directed graph

- length: number of edges on the path

  Length of V_0 to V_2 is 2

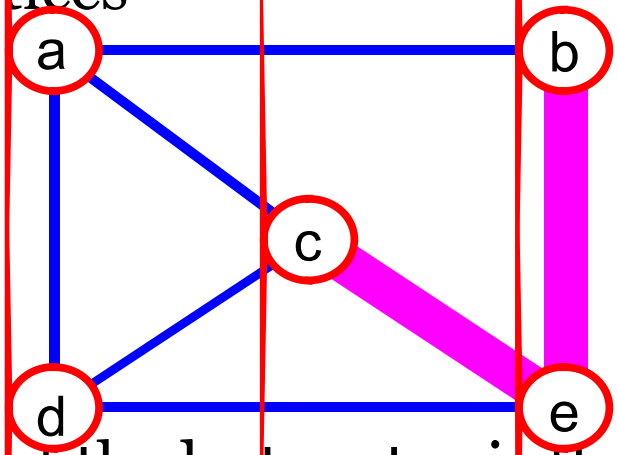
Terminology: Path



- Path: Sequence of vertices $v_1, v_2, \ldots v_k$ such that consecutive vertices v_i and v_{i+1} are adjacent.

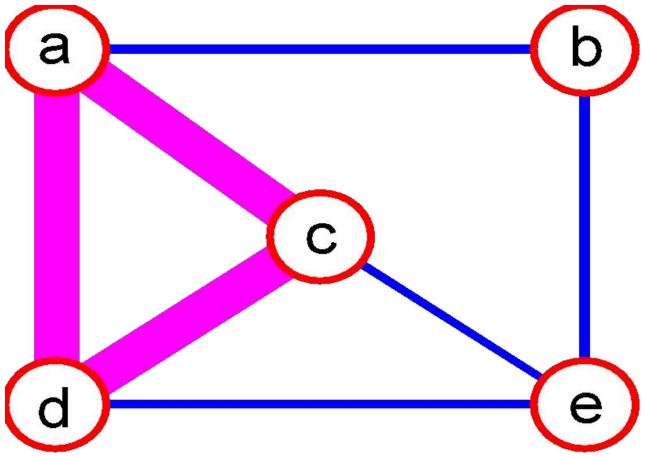
More Terminology

- simple path: no repeated vertices

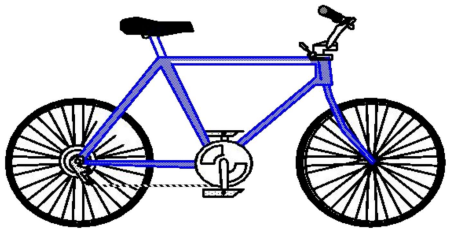


b e
c

- cycle: simple path, except that the last vertex is the same as the first vertex



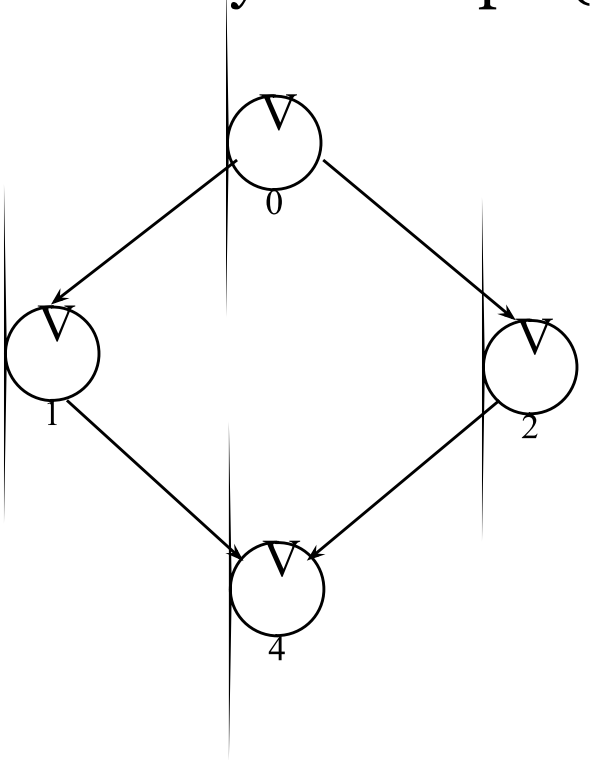
a c d a



Terminology:

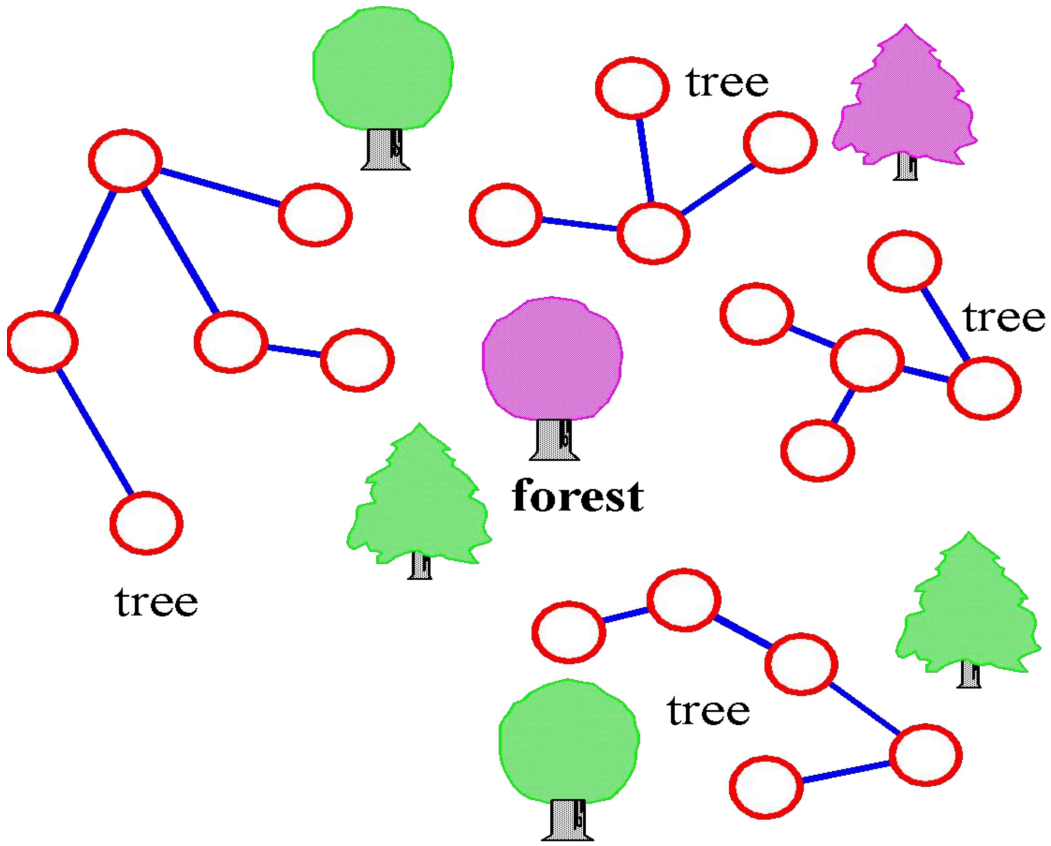
- DAG:

  A directed graph which has no cycles is known as Directed Acyclic Graph (DAG)



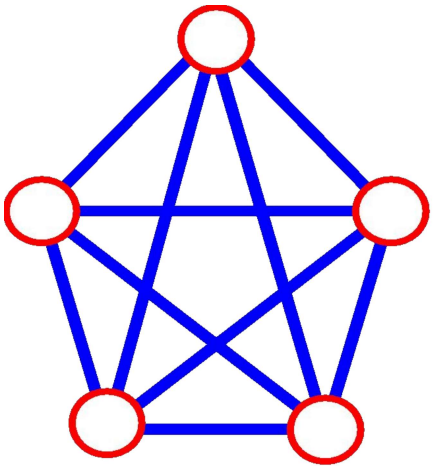
More...

- tree - connected graph without cycles
- forest - collection of trees



Connectivity

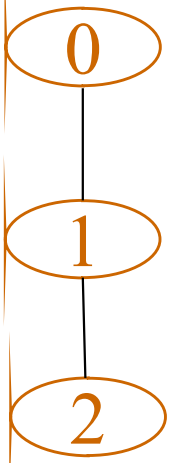
- Let **n** = #vertices, and **m** = #edges
- **A complete graph** is a graph in which there is an edge between every pair of vertices (i.e., all pairs of vertices are adjacent)
- *How many total edges in a complete graph?*
  - Each of the n vertices is incident to **n-1** edges, however, we would have counted each edge twice! Therefore, intuitively, m = **n**(**n** -1)/2.
- Therefore, if a graph is not complete, m < **n**(**n** -1)/2



n = 5
m = (5 ∗ 4)/2 = 10

Connectivity

- A graph is **connected** when there is a path between every pair of vertices



- A graph that is not connected is **disconnected**.

- A graph with just one vertex is connected.

Connectivity

- A directed graph is called ***strongly connected*** if it contains directed path from u to v and a directed path from v to u for every pair of vertices u, v.

- A directed graph is called ***weakly connected*** if it contains a directed path from u to v or a directed path from v to u for every pair of vertices u, v.

Graph Representations

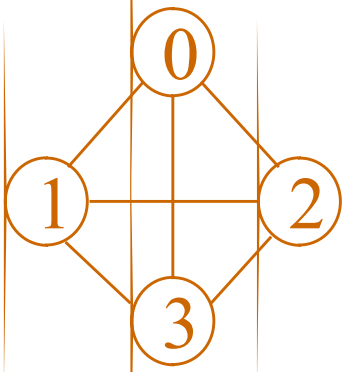
- Adjacency Matrix
- Adjacency Lists

Adjacency Matrix

- Let G=(V,E) be a graph with n vertices.
- The adjacency matrix of G is a two-dimensional n by n array, say adj_mat
- If the edge (v_i, v_j) is in E(G), adj_mat[i][j]=1
- If there is no such edge in E(G), adj_mat[i][j]=0
- The adjacency matrix for an undirected graph is symmetric; the adjacency matrix for a digraph need not be symmetric

Examples for Adjacency Matrix

G_1



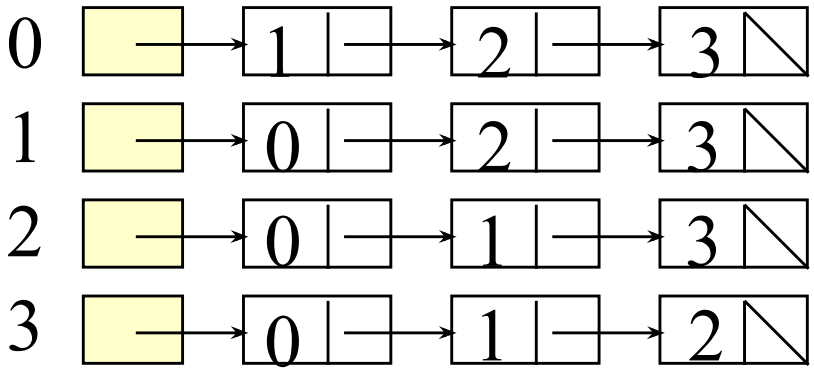
$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

(symmetric)

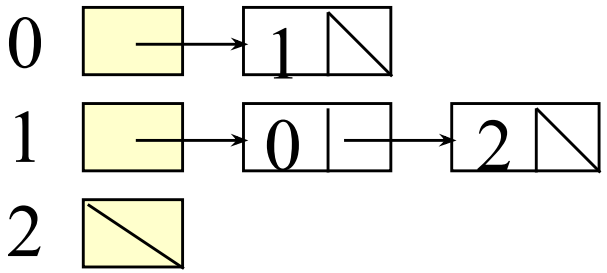
G_2

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

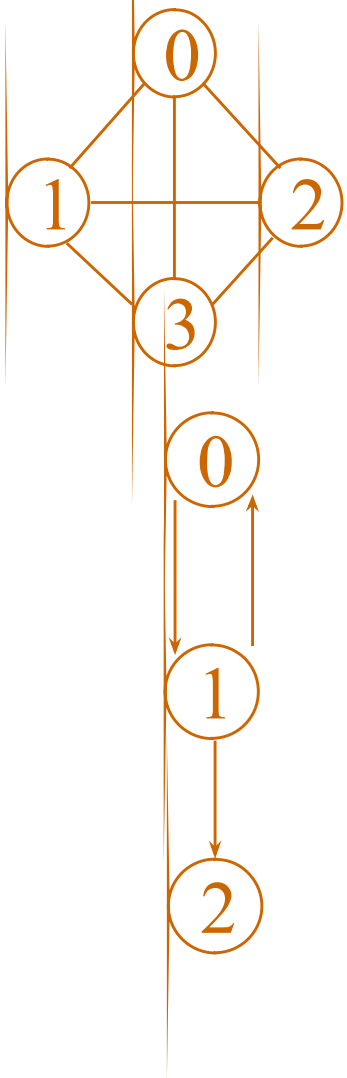
Examples for Adjacency List



G_1



G_3



Graph Traversal / Types of Graph

- Visit all the nodes in the graph exactly once
  - **Depth First Search (DFS)**
  - **Breadth First Search (BFS)**

Graph Traversal / Types of Graph

- Depth First Search (DFS)
  - Once a possible path is found, continue the search until the end of the path
- Breadth First Search (BFS)
  - Start several paths at a time, and advance in each one step at a time

Graph Traversal / Types of Graph

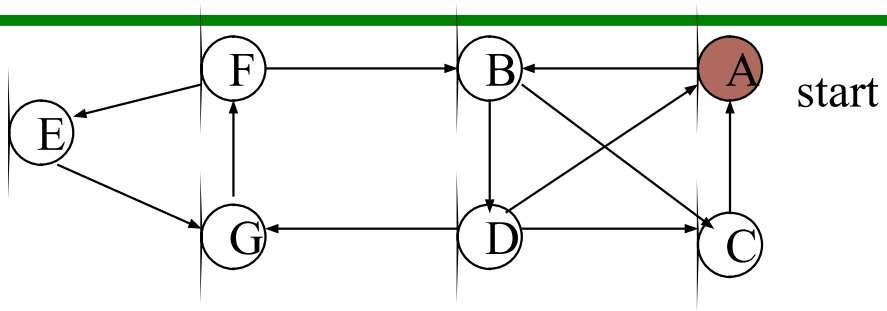
STATUS	STATE OF THE NODE	DESCRIPTION
1	Ready	The initial state of the node N
2	Waiting	Node N is placed on the queue or stack and waiting to be processed
3	Processed	Node N has been completely processed

Breadth First Search (BFS)

- Breadth-first search (BFS) is a graph search algorithm that begins at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, the algorithm explores their unexplored neighbor nodes, and so on, until it finds the goal.

BFS

BFS Process



rear	front
	A

Initial call to BFS on A
Enqueue A

rear	front
	B

Dequeue A
Enqueue B

rear	front
	D C

Dequeue B
Enqueue C, D

rear	front
	D

Dequeue C
Nothing to add

rear	front
	G

Dequeue D
Enqueue G

rear	front
	F

Dequeue G
Enqueue F

rear	front
	E

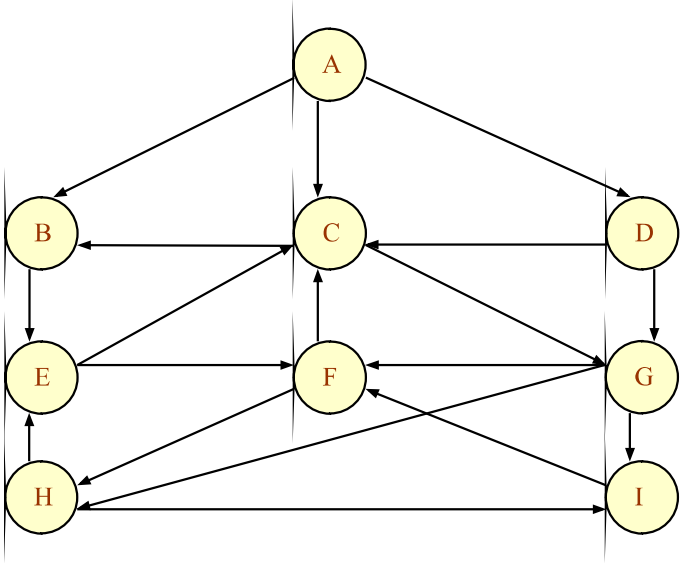
Dequeue F
Enqueue E

rear	front

Dequeue E
Nothing to add

PATH : A B C D G F E

BFS



A	B	C	D	E	G	F	H	I

H	E	I	C	F	B	G

BFS

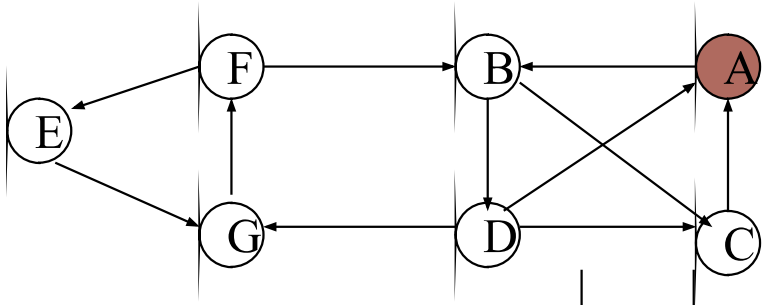
Algorithm for breadth-first search in a graph G beginning at a starting node A

Step 1: SET STATUS = 1 (ready state) for each node in G.
Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)
Step 3: Repeat Steps 4 and 5 until QUEUE is empty
Step 4: Dequeue a node N. Process it and set its STATUS = 3
(processed state).
Step 5: Enqueue all the neighbors of N that are in the STATUS = 1 (ready state) and set their STATUS = 2 (waiting state)
[END OF LOOP]
Step 6: EXIT

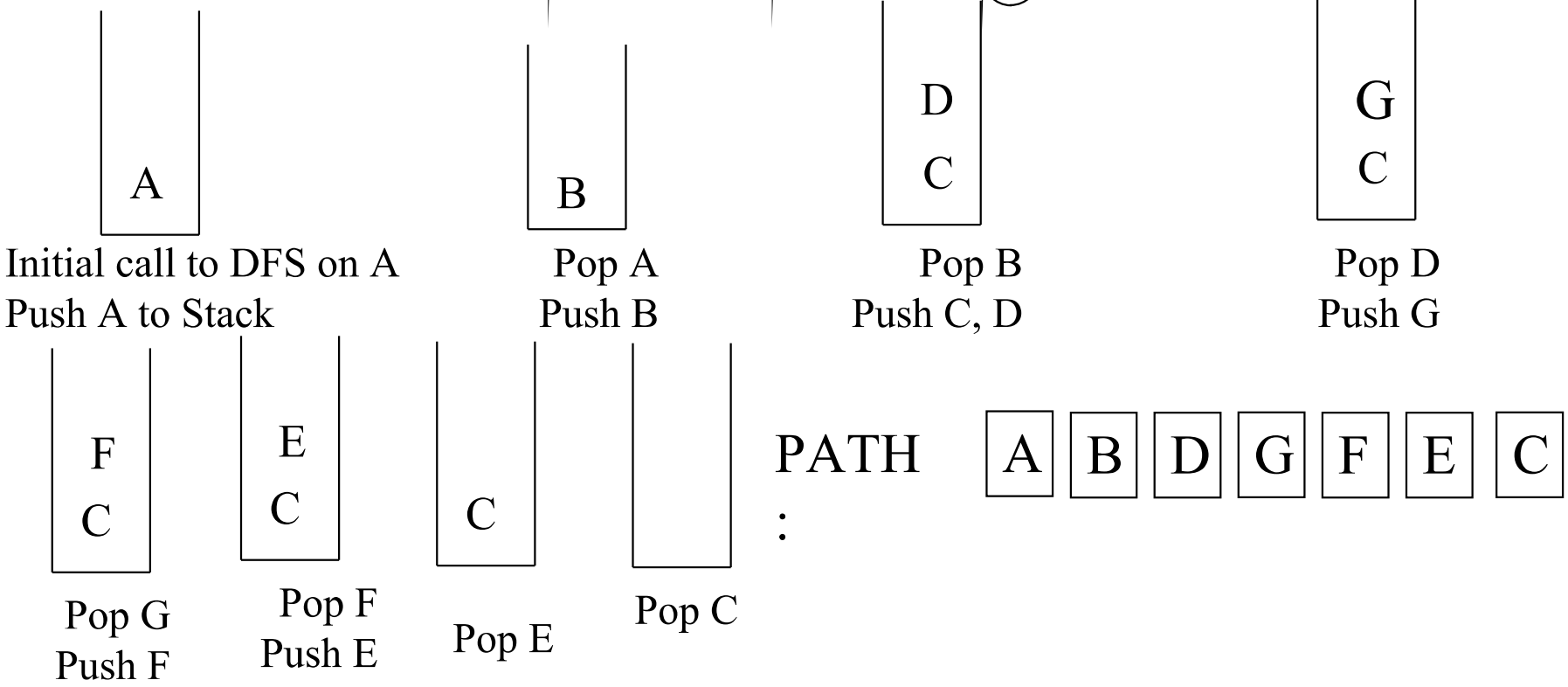
Depth First Search (DFS)

- The Depth First Search algorithm progresses by expanding the starting node of G and thus going deeper and deeper until a goal node is found, or until a node that has no children is encountered. When a dead- end is reached, the algorithm backtracks, returning to the most recent node that has not been completely explored.

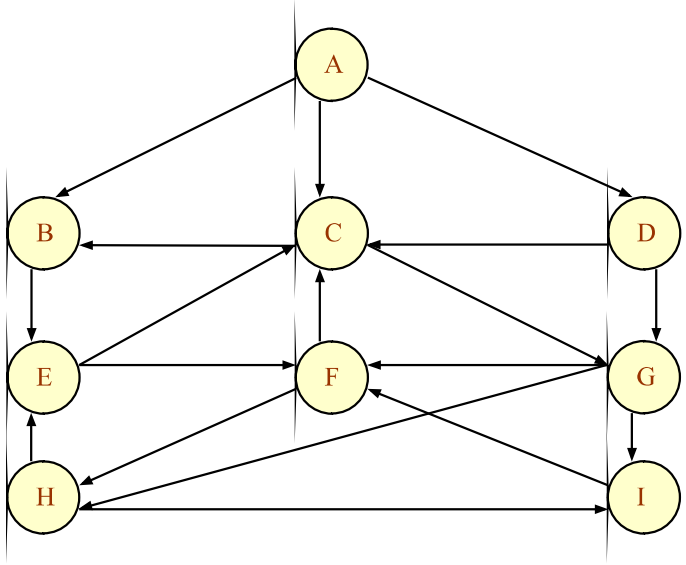
DFS



DFS Process



DFS



A	D	G	I	H	E	F	C	B

H	I	F	C	G	B	E

DFS

Algorithm for depth-first search in a graph G beginning at a starting node A

Step 1: SET STATUS = 1 (ready state) for each node in G.
Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)
Step 3: Repeat Steps 4 and 5 until STACK is empty
Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state).
Step 5: Push on to the stack all the neighbours of N that are in the STATUS = 1 (ready state) and set their STATUS = 2 (waiting state)
[END OF LOOP]
Step 6: EXIT