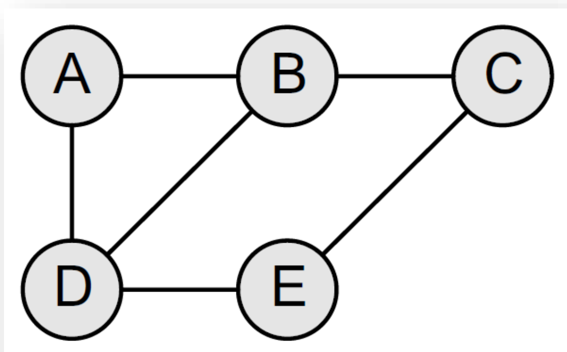
Advanced Graphs

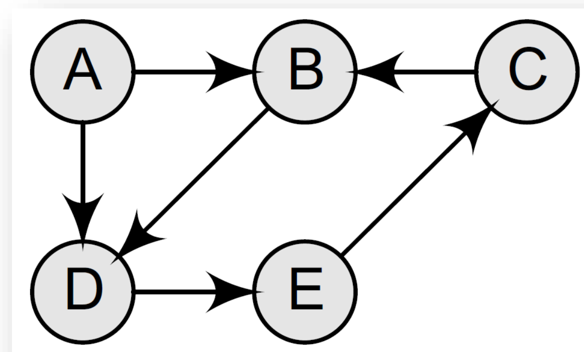
Kuan-Yu Chen (陳冠宇)

2018/12/03 @ TR-212, NTUST

Review

- A graph G is defined as an ordered set (V, E), where $V(G)$ represents the set of vertices and $E(G)$ represents the edges
  - For a given undirected graph with $V(G) = \{A, B, C, D, E\}$ and $E(G) = \{(A, B), (B, C), (A, D), (B, D), (D, E), (C, E)\}$
    - Five vertices or nodes and six edges in the graph

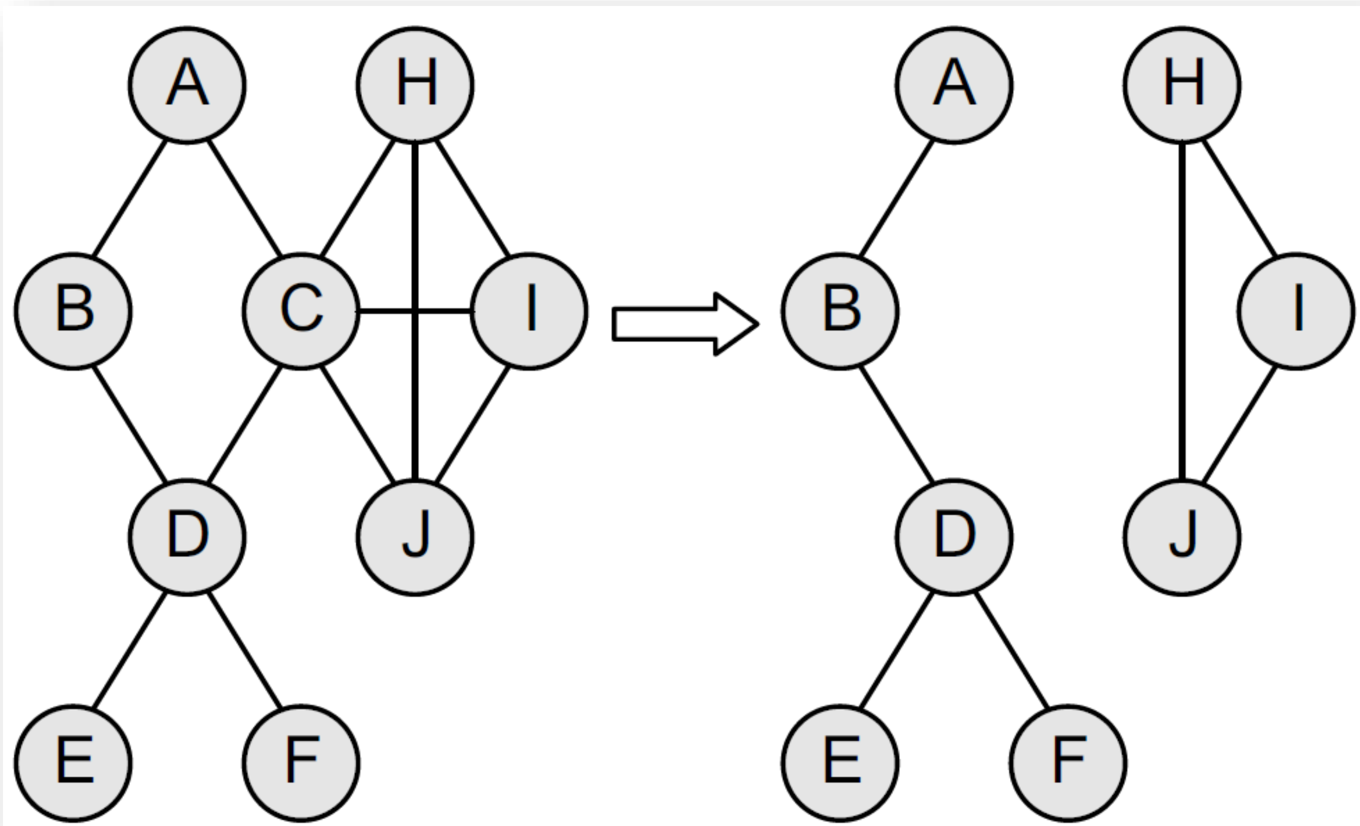




  - For a given directed graph, the edge (A, B) is said to initiate from node A (also known as initial node) and terminate at node B (terminal node)

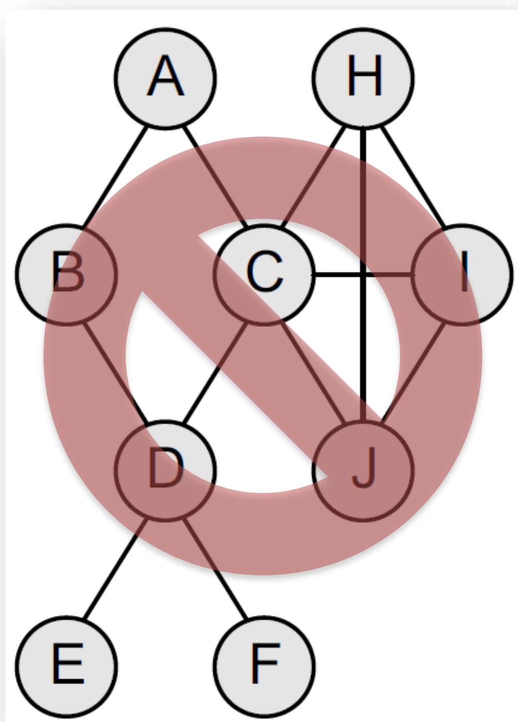
Bi-connected Components.

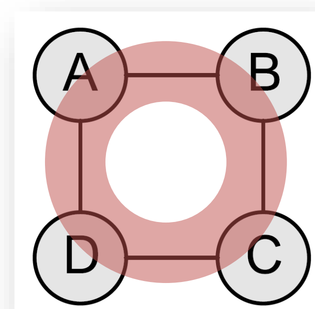
- **Articulation Point**
  - A vertex v of G is called an articulation point, if removing v along with the edges incident on v, results in a graph that has at least two connected components



Bi-connected Components..

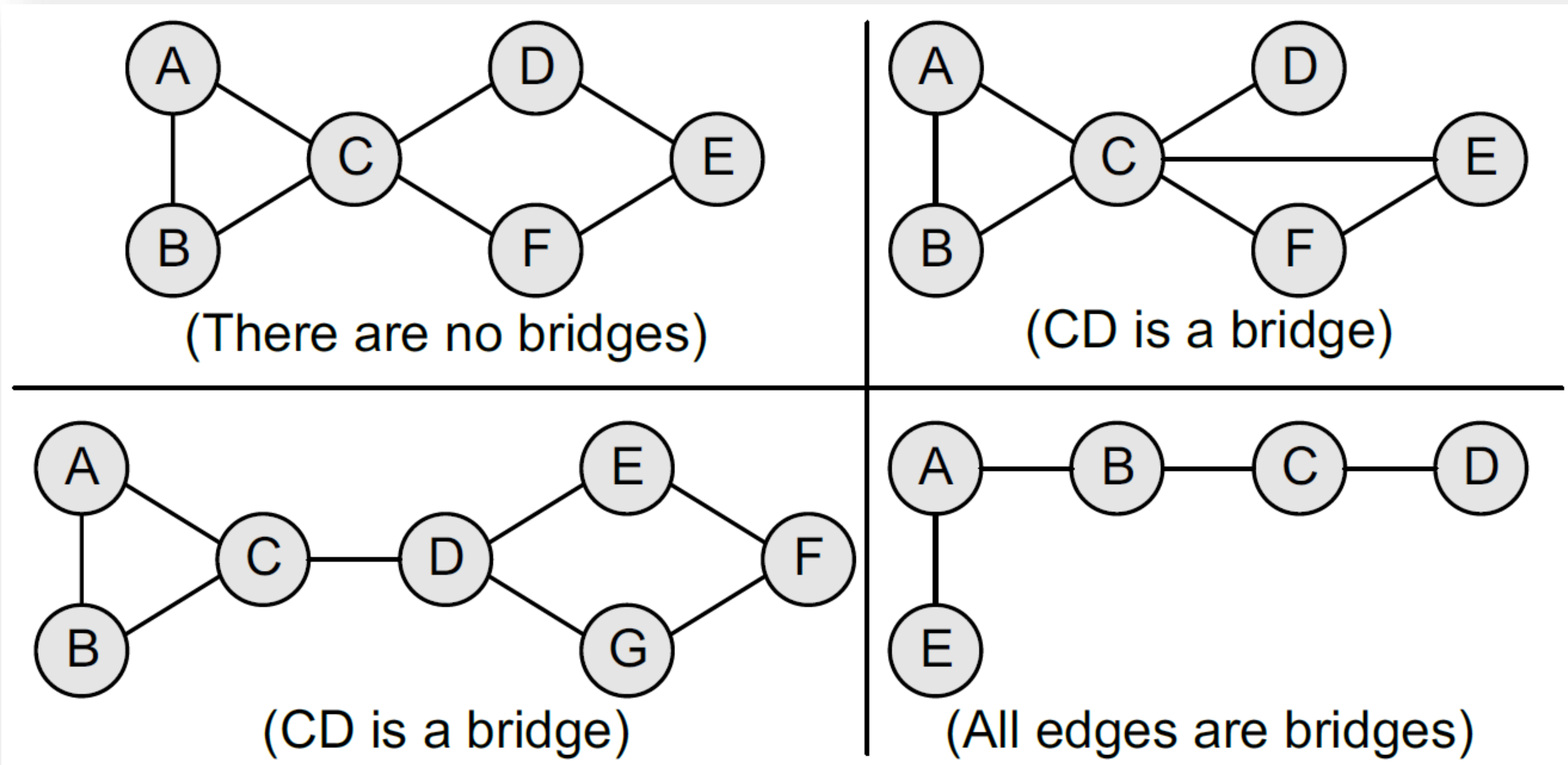
- A **bi-connected graph** is defined as a connected graph that has no articulation vertices
  - In other words, a bi-connected graph is connected and non-separable in the sense that even if we remove any vertex from the graph, the resultant graph is still connected





Bridge

- An edge in a graph is called a **bridge** if removing that edge results in a disconnected graph

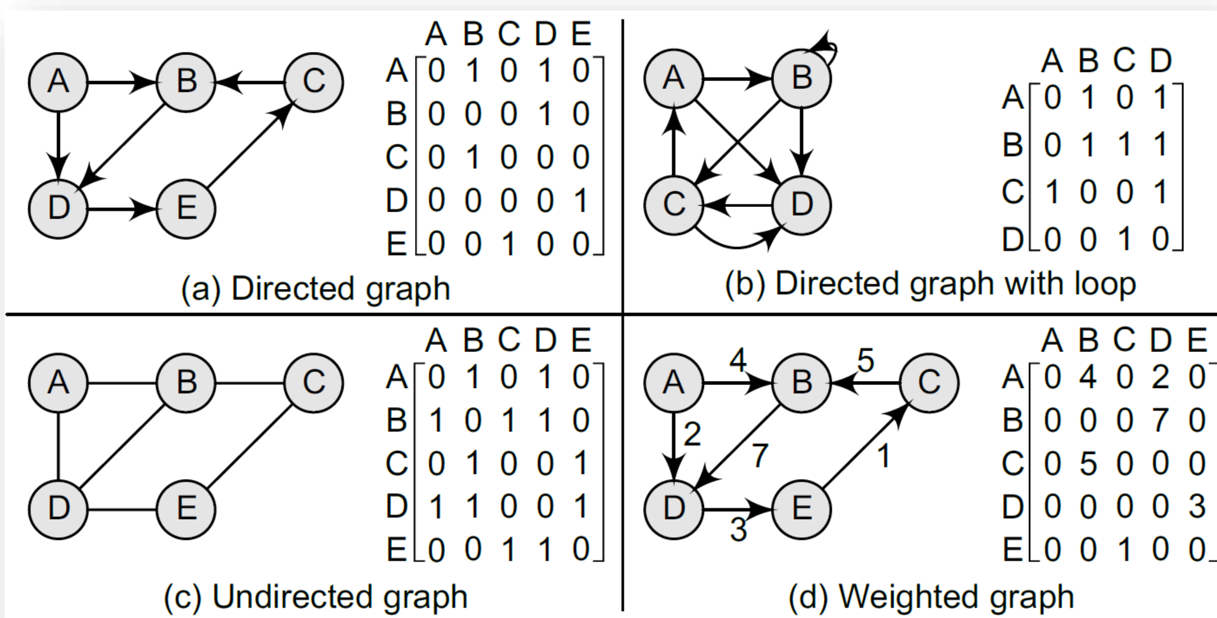


Representation of Graphs

- There are three common ways of storing graphs in the computer's memory
  - **Sequential representation** by using an adjacency matrix
  - **Linked representation** by using an adjacency list that stores the neighbors of a node using a linked list
  - **Adjacency multi-list** which is an extension of linked representation

Sequential Representation.

- For any graph G having n nodes, the adjacency matrix will have the dimension of $n \times n$
  - The rows and columns are labelled by graph vertices
  - An entry a_{ij} in the adjacency matrix will contain 1, if vertices v_i and v_j are adjacent to each other; otherwise, a_{ij} will set to 0
    - Since an adjacency matrix contains only 0s and 1s, it is called a ***bit matrix*** or a ***Boolean matrix***



(a) Directed graph

(b) Directed graph with loop

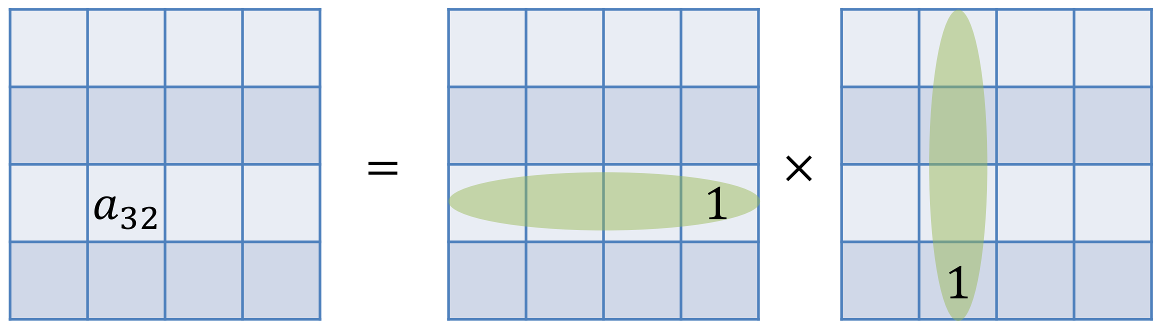
(c) Undirected graph

(d) Weighted graph

Sequential Representation..

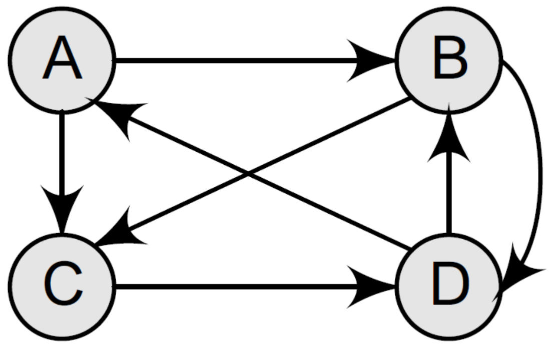
- From the original adjacency matrix, denoted by A^1
  - An entry 1 in the i^{th} row and j^{th} column means that there exists a path of length 1 from v_i to v_j

- Let's consider A^2
  - $A^2 = A^1 \times A^1$
  - $a_{ij}^2 = \sum a_{ik} a_{kj}$
  - If $a_{ij}^2 \geq 1$, $\exists k$ such that $a_{ik} = 1 \ \wedge a_{kj} = 1$
  - That is, if there is an edge (v_i, v_k) and $\left(v_k, v_j\right)$, then there is a path from v_i to v_j of length 2



- Similarly, every entry in the i^{th} row and j^{th} column of A^n gives the number of paths of length n from node v_i to v_j

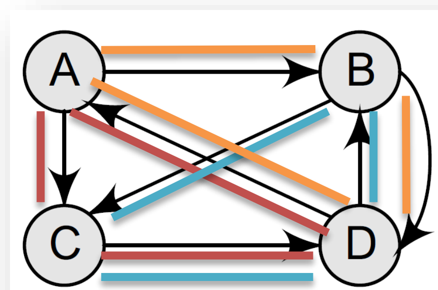
Sequential Representation…



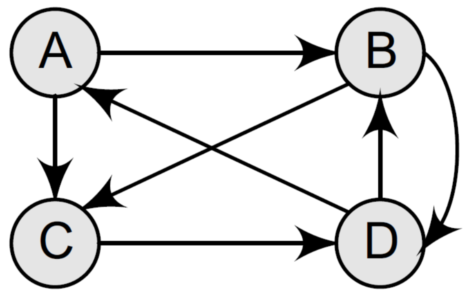
$$
\begin{array}{c|cccc}
 & A & B & C & D \\
\hline
A & 0 & 1 & 1 & 0 \\
B & 0 & 0 & 1 & 1 \\
C & 0 & 0 & 0 & 1 \\
D & 1 & 1 & 0 & 0
\end{array}
$$

– $A^2 = A^1 \times A^1 = \begin{bmatrix} 0012 \\ 1101 \\ 1100 \\ 0121 \end{bmatrix}$

– $A^3 = A^2 \times A^1 = \begin{bmatrix} 2201 \\ 1221 \\ 0121 \\ 1113 \end{bmatrix}$



Sequential Representation….



$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

– We can further define a matrix $B^n = A^1 + \cdots + A^n$

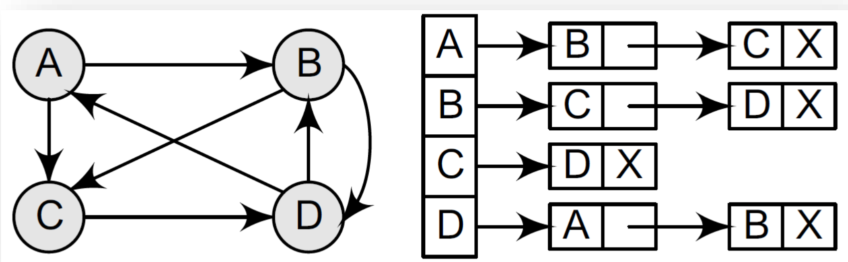
- $B^3 = A^1 + A^2 + A^3 = \begin{bmatrix} 0110 \\ 0011 \\ 0001 \\ 1100 \end{bmatrix} + \begin{bmatrix} 0012 \\ 1101 \\ 1100 \\ 0121 \end{bmatrix} + \begin{bmatrix} 2201 \\ 1221 \\ 0121 \\ 1113 \end{bmatrix} = \begin{bmatrix} 2323 \\ 2333 \\ 1222 \\ 2334 \end{bmatrix}$

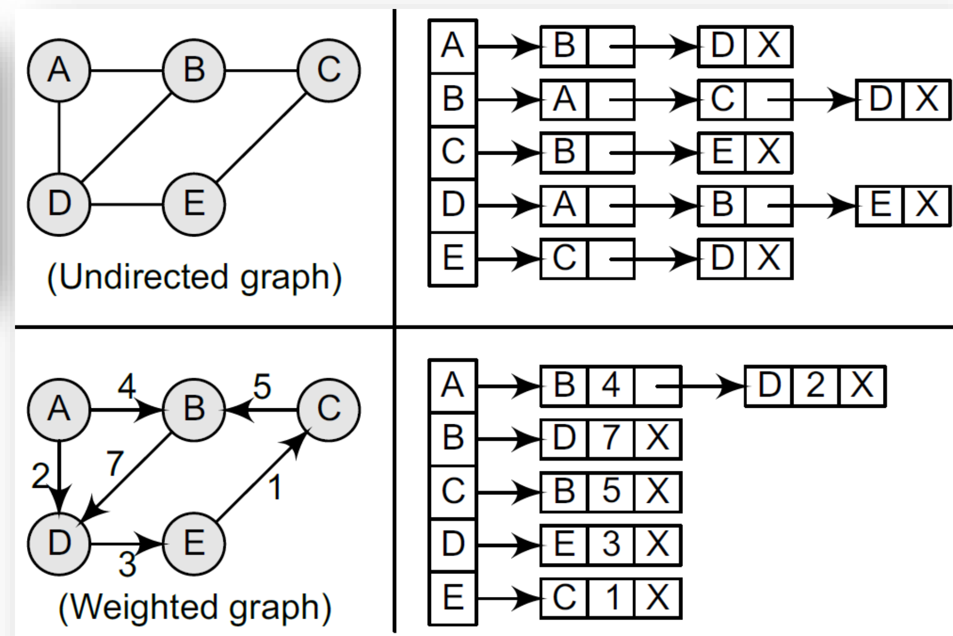
– A path matrix P can be obtained by setting an entry $p_{ij} = 1$, if b_{ij} is non-zero and $p_{ij} = 0$, if otherwise

- $P = \begin{bmatrix} 1111 \\ 1111 \\ 1111 \\ 1111 \end{bmatrix}$

Linked Representation

- An adjacency list is another way in which graphs can be represented in the computer's memory
  - It is often used for storing graphs that have a small-to-moderate number of edges
    - That is, an adjacency list is preferred for representing **sparse graphs** in the computer's memory; otherwise, an adjacency matrix is a good choice



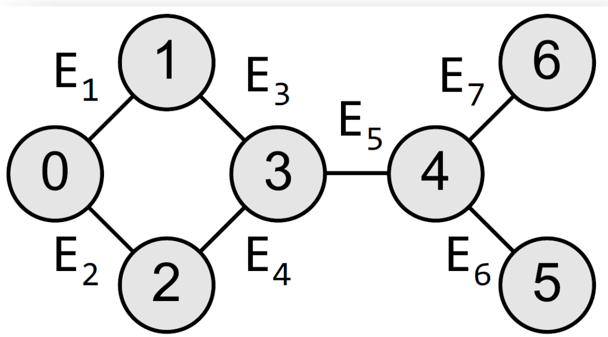


(Undirected graph)

(Weighted graph)

Adjacency Multi-list.

- Graphs can also be represented using multi-lists which can be said to be modified version of adjacency lists
  - Adjacency multi-list is an **edge-based** rather than a **vertex-based** representation of graphs



Edge 1		0	1	Edge 2	Edge 3
Edge 2		0	2	NULL	Edge 4
Edge 3		1	3	NULL	Edge 4
Edge 4		2	3	NULL	Edge 5
Edge 5		3	4	NULL	Edge 6
Edge 6		4	5	Edge 7	NULL
Edge 7		4	6	NULL	NULL

Adjacency Multi-list..

Edge 1		0	1	Edge 2	Edge 3
Edge 2		0	2	NULL	Edge 4
Edge 3		1	3	NULL	Edge 4
Edge 4		2	3	NULL	Edge 5
Edge 5		3	4	NULL	Ed
Edge 6		4	5	Edge 7	NU
Edge 7		4	6	NULL	NU

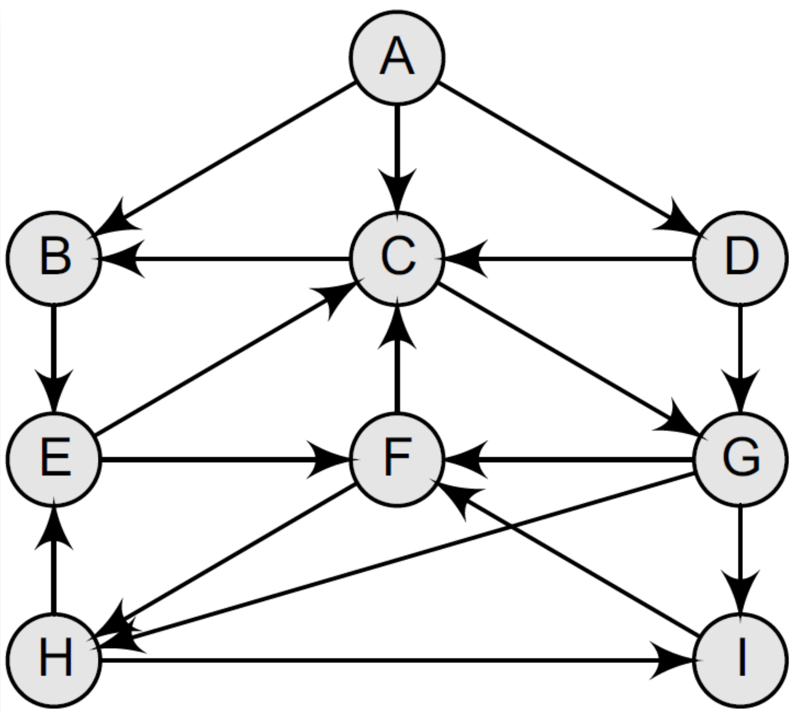
VERTEX	LIST OF EDGES
0	Edge 1, Edge 2
1	Edge 1, Edge 3
2	Edge 2, Edge 4
3	Edge 3, Edge 4, Edge 5
4	Edge 5, Edge 6, Edge 7
5	Edge 6
6	Edge 7

Traversal Algorithms

- By traversing a graph, we mean the method of examining the nodes and edges of the graph
  - Breadth-first search
    - BFS uses a **queue** as an auxiliary data structure to store nodes for further processing
  - Depth-first search
    - DFS uses a **stack** to store nodes for further processing

Breadth-first Search.

- Breadth-first search (BFS) is a graph search algorithm that begins at the root node and explores all the neighboring nodes
  - Given a directed graph, please find a minimum path from *A* to *I* by using BFS

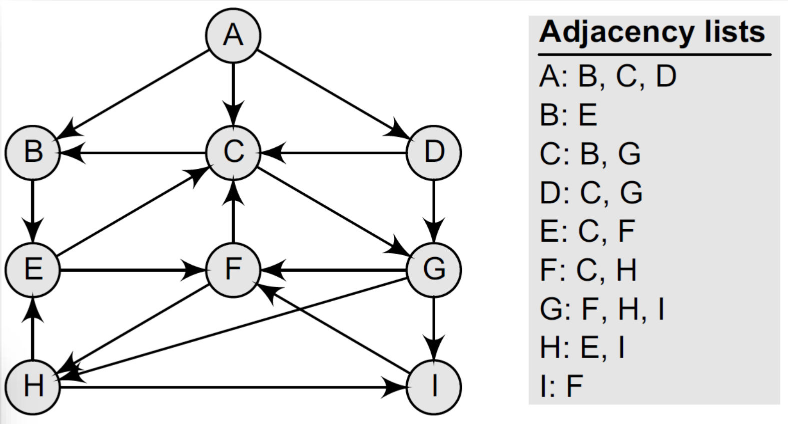


Adjacency lists

A: B, C, D
B: E
C: B, G
D: C, G
E: C, F
F: C, H
G: F, H, I
H: E, I
I: F

Breadth-first Search..

- **QUEUE** is used to hold the nodes that have to be processed, **ORIG** is used to keep track of the origin of each edge
  - Step 1:



QUEUE	=	A
ORIG	=	\0

  - Step 2:

QUEUE	=	A	B	C	D
ORIG	=	\0	A	A	A

  - Step 3:

QUEUE	=	A	B	C	D	E
ORIG	=	\0	A	A	A	B

  - Step 4:

QUEUE	=	A	B	C	D	E	G
ORIG	=	\0	A	A	A	B	C

Breadth-first Search...

– Step 5:

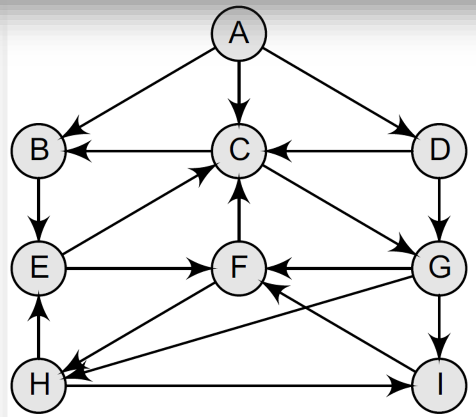
QUEUE =	A	B	C	D	E	G
ORIG =	\0	A	A	A	B	C

– Step 6:

QUEUE =	A	B	C	D	E	G	F
ORIG =	\0	A	A	A	B	C	E

– Step 7:

QUEUE =	A	B	C	D	E	G	F	H	I
ORIG =	\0	A	A	A	B	C	E	G	G



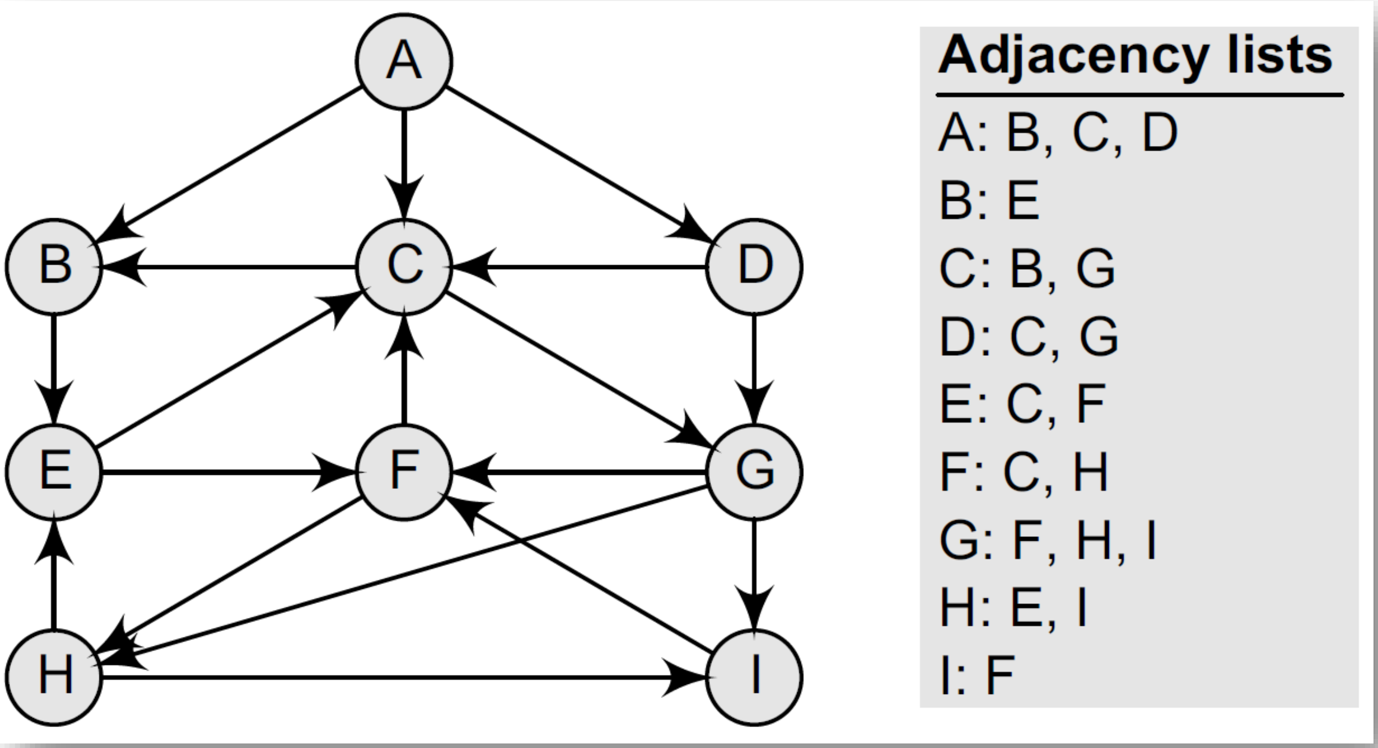
Adjacency lists

A: B, C, D
B: E
C: B, G
D: C, G
E: C, F
F: C, H
G: F, H, I
H: E, I
I: F

Breadth-first Search....

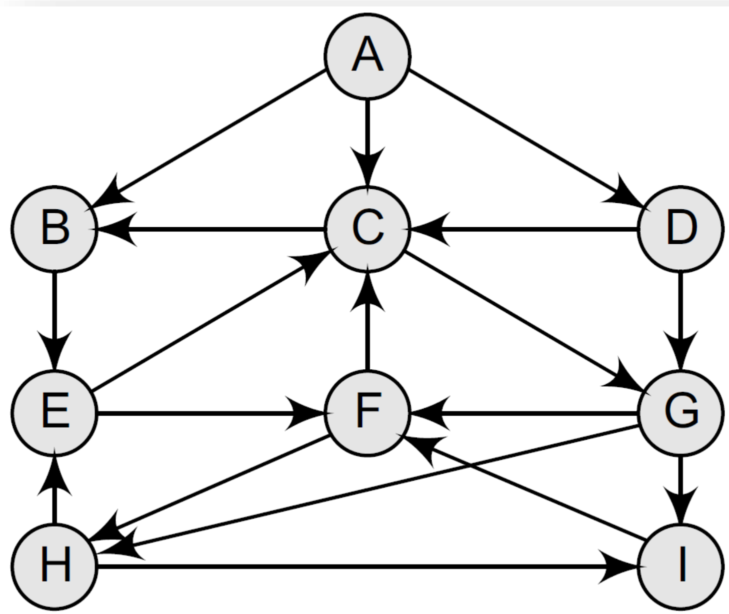
– Final, by referring to ORIG, the minimum path is $A \to C \to G \to I$

QUEUE =	A	B	C	D	E	G	F	H	I
ORIG =	\0	A	A	A	B	C	E	G	G



Depth-first Search.

- The depth-first search algorithm progresses by expanding the starting node of G and then going deeper and deeper until the goal node is found, or until a node that has no children is encountered
  - Given a graph G and its adjacency list, please print all the nodes that can be reached from the node H (including H itself) by leveraging DFS



Adjacency lists

A: B, C, D
B: E
C: B, G
D: C, G
E: C, F
F: C, H
G: F, H, I
H: E, I
I: F

Depth-first Search..

- Step 1: Push *H* onto the stack

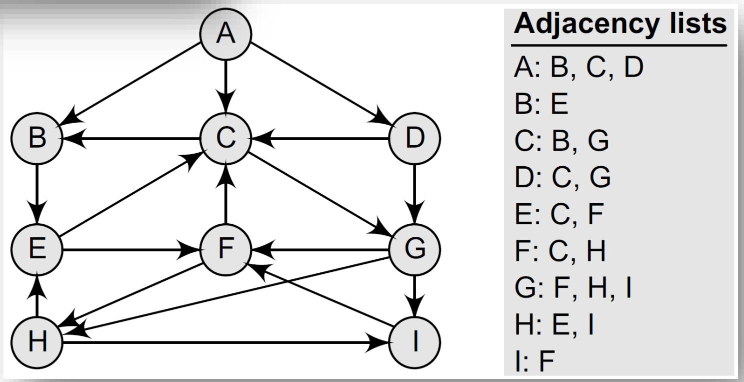
STACK: H

- Step 2:
  - Pop and print the top element of the stack (i.e., *H*)
  - Push all the neighbors of *H* onto the stack

PRINT: H   STACK: E, I

- Step 3:
  - Pop and print the top element of the stack (i.e., *I*)
  - Push all the neighbors of *I* onto the stack

PRINT: I   STACK: E, F



Depth-first Search...

- Step 4:
  - Pop and print the top element of the stack (i.e., F)
  - Push all the neighbors of F onto the stack

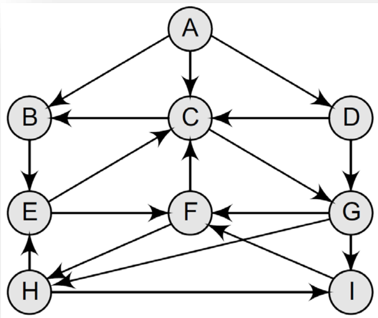
PRINT: F	STACK: E, C

- Step 5:
  - Pop and print the top element of the stack (i.e., C)
  - Push all the neighbors of C onto the stack

PRINT: C	STACK: E, B, G

- Step 6:
  - Pop and print the top element of the stack (i.e., G)
  - Push all the neighbors of G onto the stack

PRINT: G	STACK: E, B



Adjacency lists
A: B, C, D
B: E
C: B, G
D: C, G
E: C, F
F: C, H
G: F, H, I
H: E, I
I: F

Depth-first Search….

- Step 7:
  - Pop and print the top element of the stack (i.e., B)
  - Push all the neighbors of B onto the stack

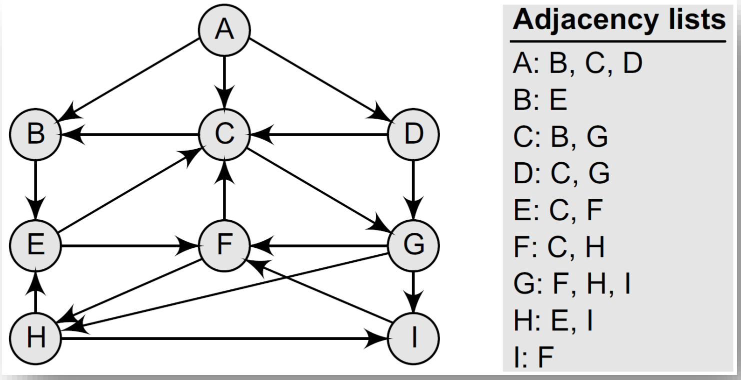
PRINT: B | STACK: E

- Step 8:
  - Pop and print the top element of the stack (i.e., E)
  - Push all the neighbors of E onto the stack

PRINT: E | STACK:

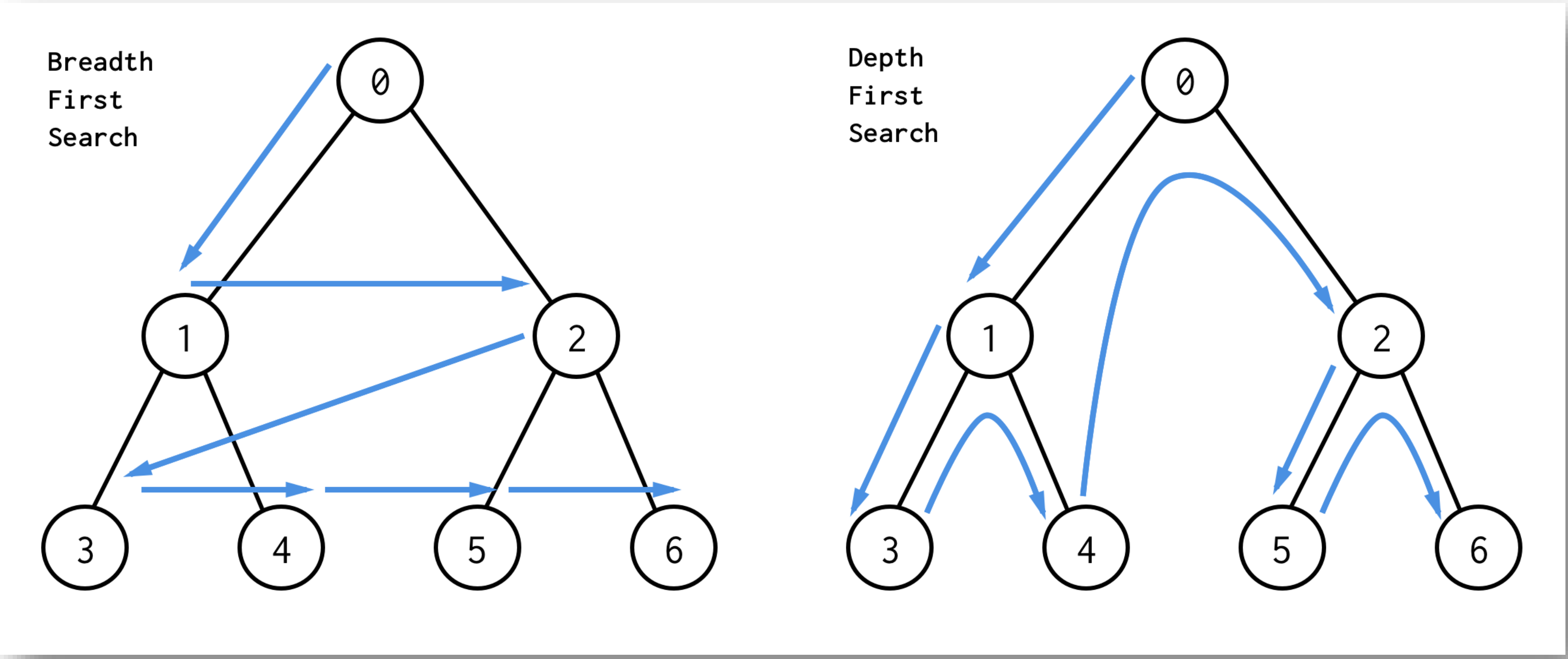
- Since the stack is empty, the depth-first search of G starting at node H is complete and the nodes which were printed are

H, I, F, C, G, B, E



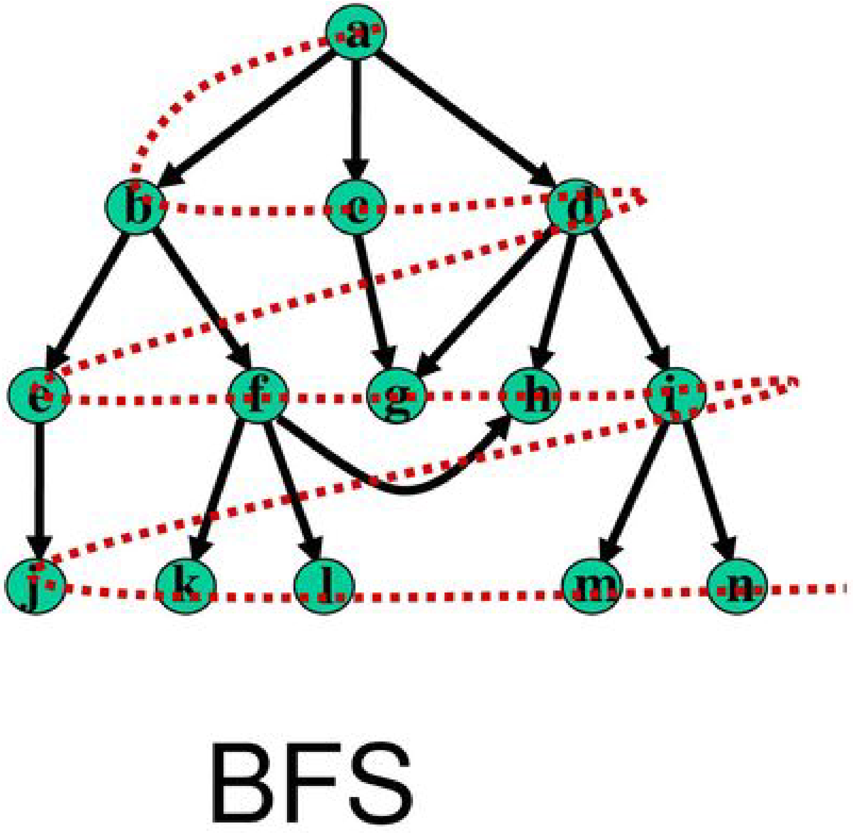
BFS & DFS.

- https://www.quora.com/What-are-the-differences-between-DFS-and-BFS

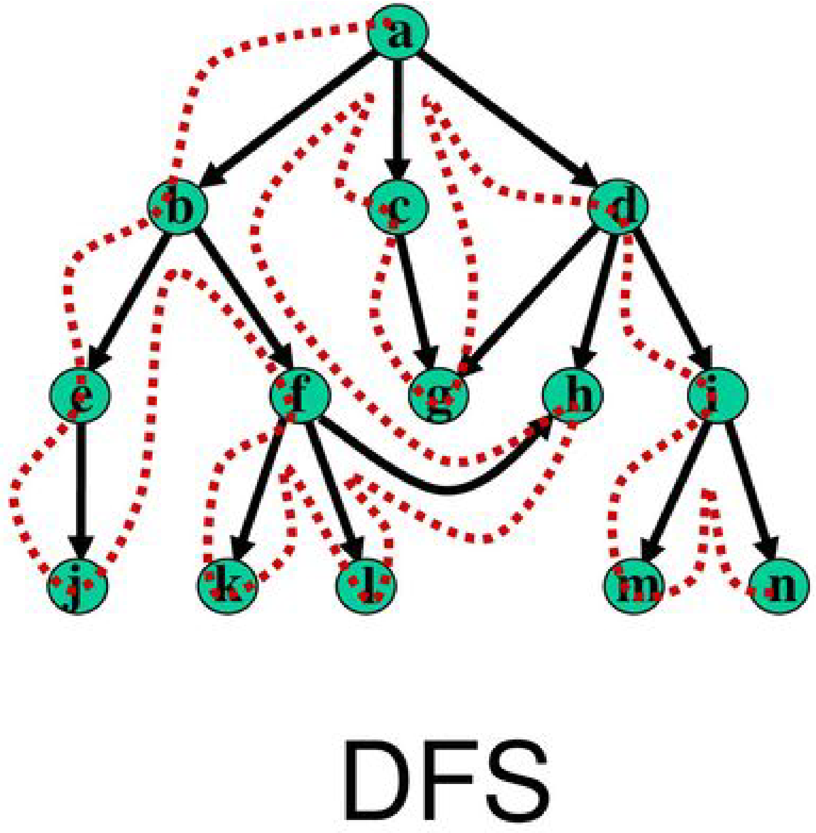


BFS & DFS..

- https://slideplayer.com/slide/12046827/



a,b,c,d,e,f,g,h,i,j,k,l,m,n



a,b,e,j,f,k,l,h,c,g,d,i,m,n

Questions?

kychen@mail.ntust.edu.tw