Graphs

Data Structures – CS2001 – Fall 2021

Dr. Syed Ali Raza

School of Computer Science

National University of Computing & Emerging Sciences

Karachi Campus

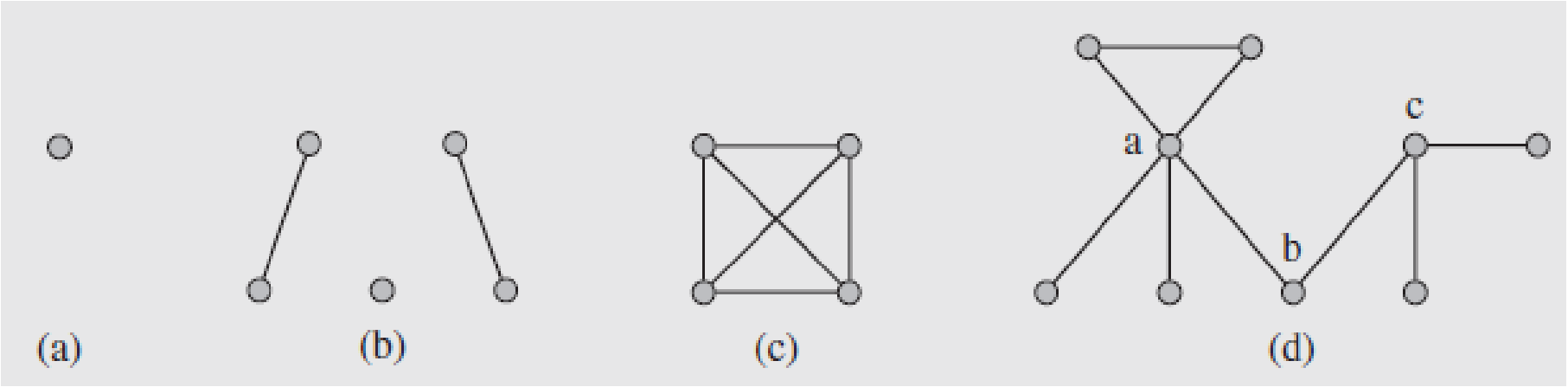
Graphs

- Trees can represent relations of a hierarchical type, such as relations between parent and child. Other relations are only represented indirectly, such as the relation of being a sibling.

- A graph is a collection of vertices (or nodes) and the connections between them.

- Formally, a *simple graph* $G = (V, E)$ consists of a nonempty set V of *vertices* and a set E of *edges* (which can be empty), each edge being a set of two vertices from V.

Graphs

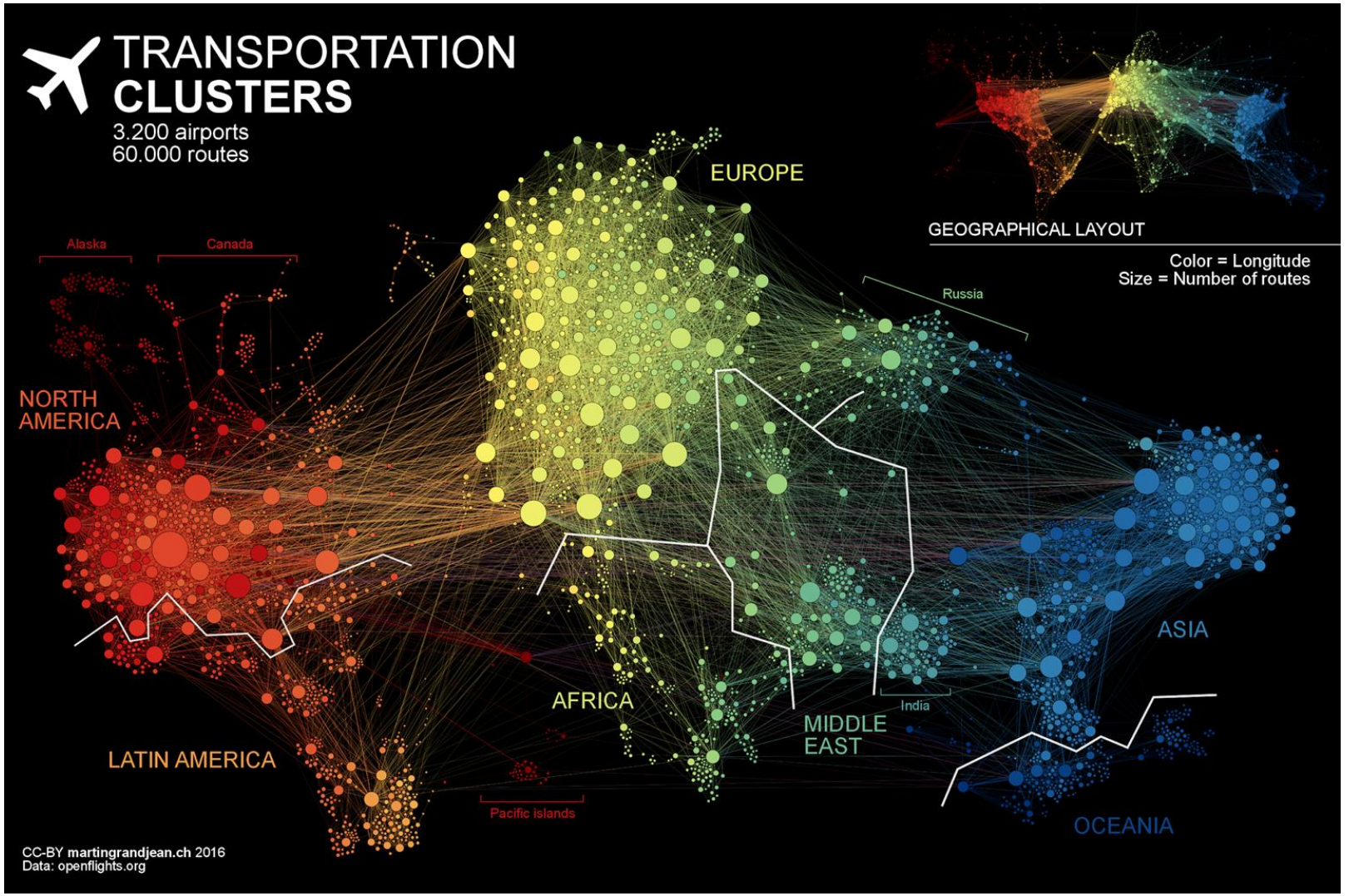


Examples of Simple Graphs

- The number of vertices and edges is denoted by $|V|$ and $|E|$, respectively.

GRAPH EXAMPLE

Each “node” is an airport, and flight routes are represented by the “edge” in between them

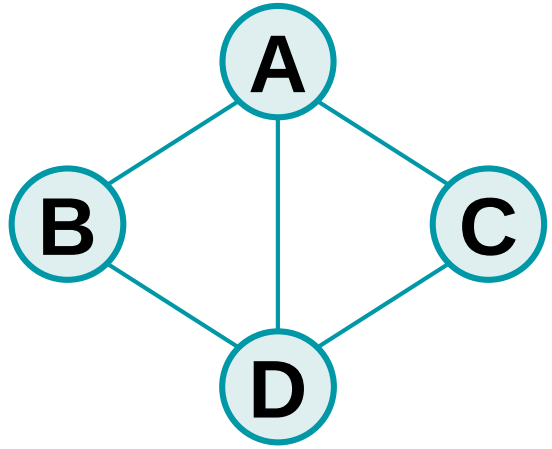


2 KINDS OF GRAPHS

UNDIRECTED GRAPHS

An undirected graph has
a set of vertices (V) & a set of edges (E)

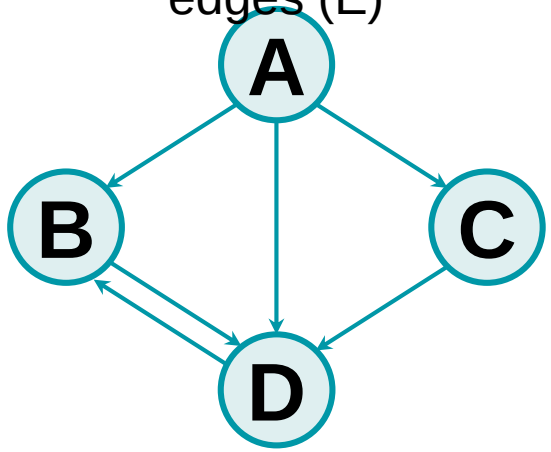
Formally,
G = (V, E)



V = {A, B, C, D}
E = { {A, B}, {A, C}, {A, D}, {B, D}, {C, D}}

DIRECTED GRAPHS

A directed graph has
a set of vertices (V) & a set of **DIRECTED** edges (E)



Formally,
G = (V, E)

V = {A, B, C, D}
E = { [A, B], [A, C], [A, D], [B, D], [C, D], [D, B]}

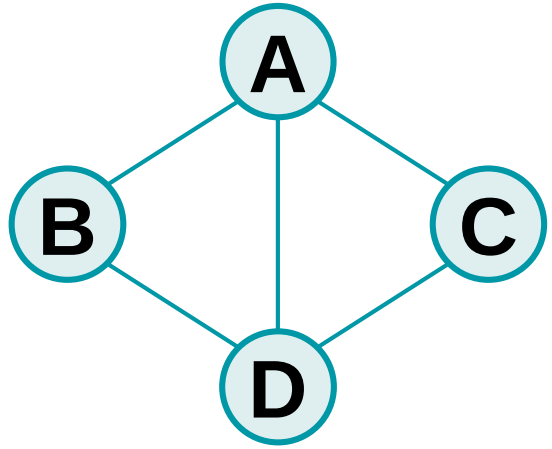
Adopted from: https://web.stanford.edu/class/cs161/

2 KINDS OF GRAPHS

UNDIRECTED GRAPHS

An undirected graph has
a set of vertices (V) & a set of edges (E)

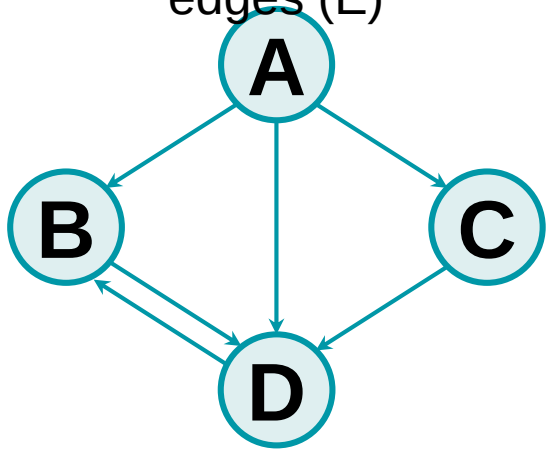
Formally,
G = (V, E)



The **degree** of vertex D is 3
Vertex D's **neighbors** are A, B, and C

DIRECTED GRAPHS

A directed graph has
a set of vertices (V) & a set of **DIRECTED** edges (E)



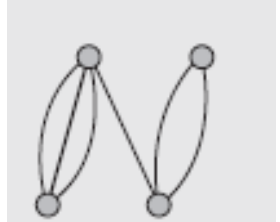
Formally,
G = (V, E)

The **in-degree** of vertex D is 3. The **out-degree** of vertex D is 1.
Vertex D's **incoming neighbors** are A, B, & C
Vertex D's **outgoing neighbor** is B

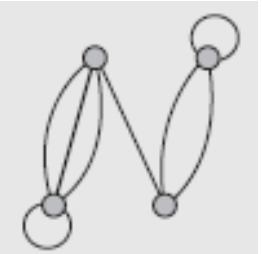
Adopted from: https://web.stanford.edu/class/cs161/

Graphs – Terminologies

- A *directed graph* is also known as a *digraph.*
- For an edge of form $\{v_i, v_j\}$ in an undirected graph, $\{v_i, v_j\} = \{v_j, v_i\}$.
- For a digraph, $\{v_i, v_j\} \neq \{v_j, v_i\}$.
- A *multigraph* is a graph in which two vertices can be joined by multiple edges. *Loops (edge from a node to itself)* are not allowed (e).
- A *pseudograph* is a multigraph where *loops* are allowed (f).



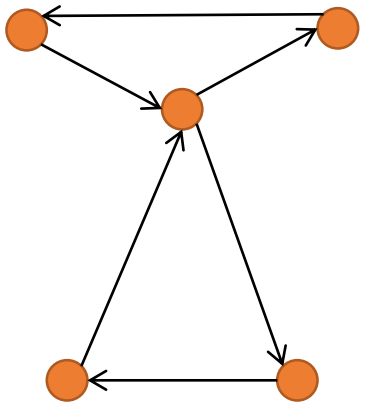
(e)



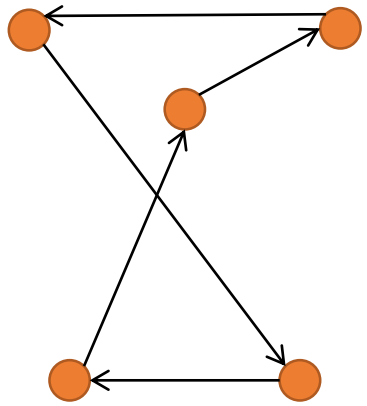
(f)

Graphs – Terminologies

- A *path* from v_1 to v_n is a sequence of edges *edge*(v_1v_2), *edge*(v_2v_3), . . . , *edge* $(v_{n-1}v_n)$ and is denoted as path $v_1, v_2, v_3, \ldots, v_{n-1}, v_n$.
- If $v_1 = v_n$ and no edge is repeated, then the path is called a *circuit* (g). If all vertices in a circuit are different, then it is called a *cycle (h).*



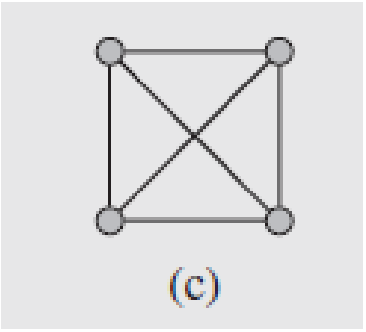
path



circuit

Graphs – Terminologies

- A graph is called a *weighted graph* if each edge has an assigned number. Depending on the context in which such graphs are used, the number assigned to an edge is called its weight, cost, distance, length, or some other name.
- A graph with *n* vertices is called *complete,* and is denoted K_n, if for each pair of distinct vertices there is exactly one edge connecting them; that is, each vertex can be connected to any other vertex (c).



K_4: 6

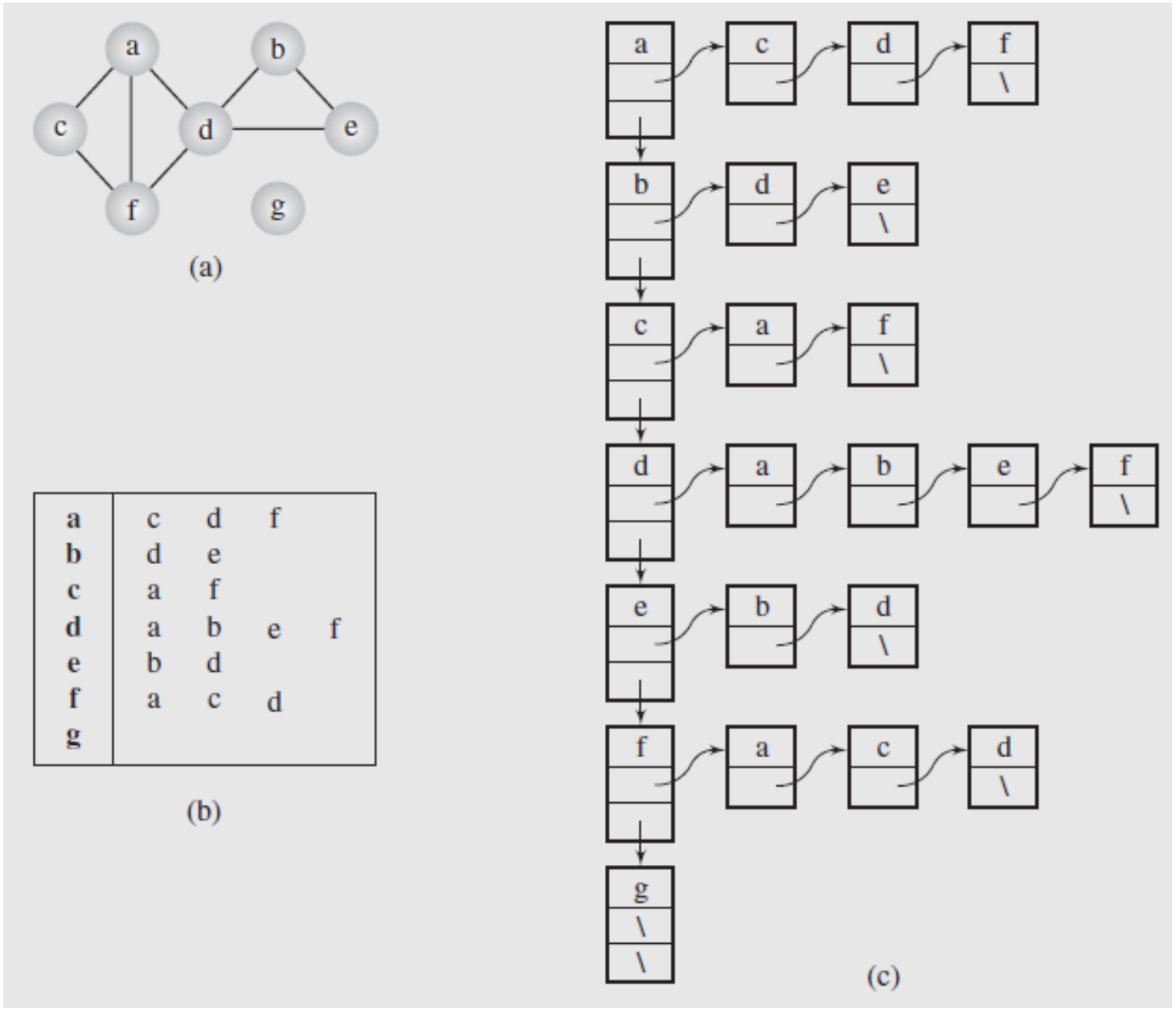
Graphs – Terminologies

- A *subgraph G′* of graph *G* = (*V,E*) is a graph (*V′*, *E′*) such that $V' \subseteq V$ and $E' \subseteq E$.
- Two vertices v_i and v_j are called *adjacent* if the *edge*($v_i v_j$) is in *E.* Such an edge is called *incident with* the vertices v_i and v_j.
- Therefore, the *degree* of a vertex *v, deg*(*v*), is the number of edges incident with *v.*
- If *deg*(*v*) = 0, then *v* is called an *isolated vertex.*
- If the set of edges *E* is empty, it means graph is consisting only of isolated vertices.

Graph Representation

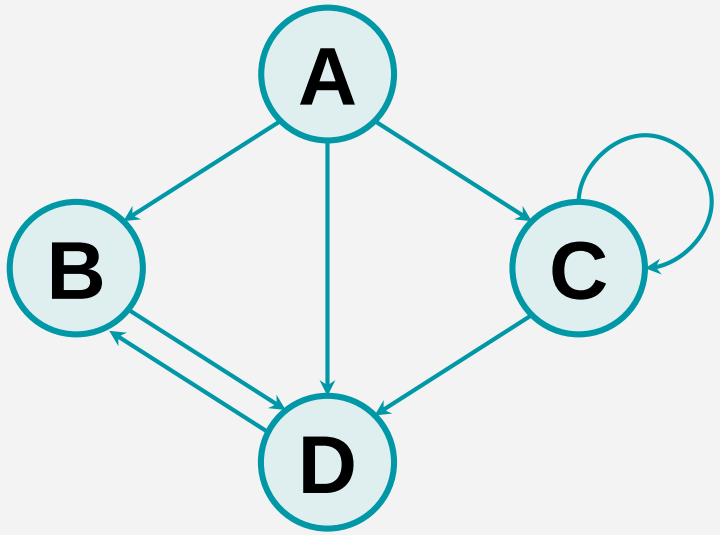
- There are various ways to represent a graph.
- A simple representation is given by an *adjacency list*, which specifies all vertices adjacent to each vertex of the graph.
- This list can be implemented as a table (Fig. b), in which case it is called a *star representation*, or as a linked list (Fig. c).

Graph Representation

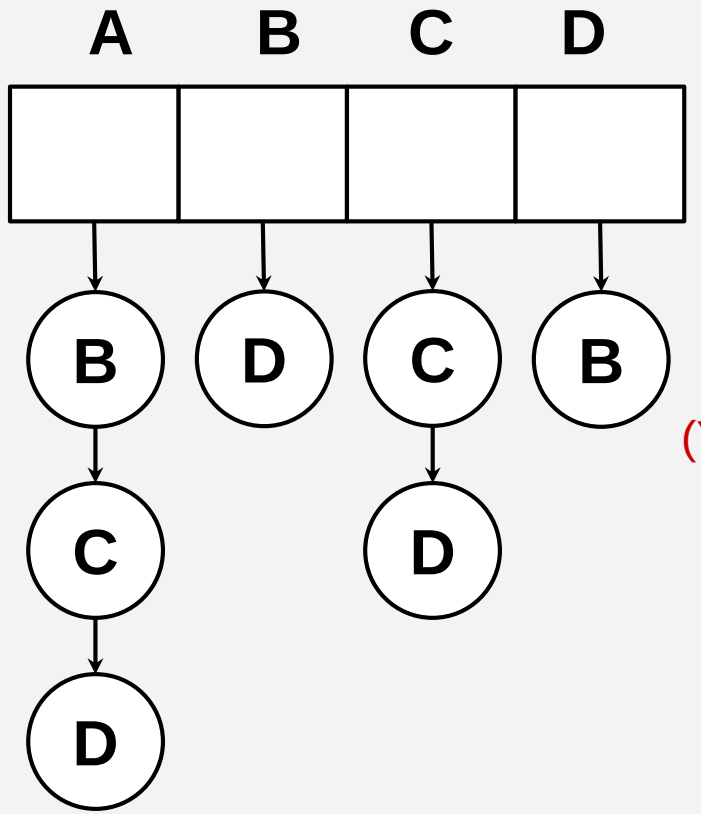


GRAPH REPRESENTATION

ADJACENCY LISTS



(A directed graph)



Tracks *outgoing neighbors*.

(You could also do the same for incoming neighbors as well)

Graph Representation

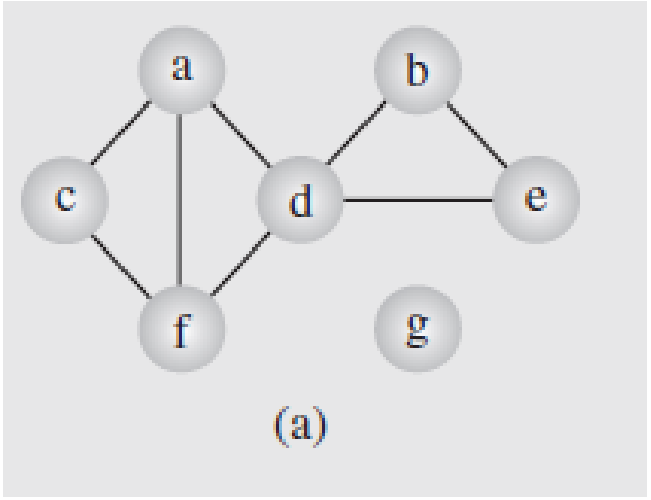
- Another representation is a matrix, which comes in two forms: an *adjacency matrix* and an *incidence matrix*.
- An *adjacency matrix* (Fig. d) of graph $G = (V,E)$ is a binary $|V| \times |V|$ matrix such that each entry, a_{ij} , of this matrix is,

$$a_{ij} = \begin{cases} 1 & \text{if there exists an edge}(v_i v_j) \\ 0 & \text{otherwise} \end{cases}$$

- An *incidence matrix* (Fig. e) of graph $G = (V,E)$ is a $|V| \times |E|$ matrix such that

$$a_{ij} = \begin{cases} 1 & \text{if edge } e_j \text{ is incident with vertex } v_i \\ 0 & \text{otherwise} \end{cases}$$

Graph Representation



(a)

	a	b	c	d	e	f	g
a	0	0	1	1	0	1	0
b	0	0	0	1	1	0	0
c	1	0	0	0	0	1	0
d	1	1	0	0	1	1	0
e	0	1	0	1	0	0	0
f	1	0	1	1	0	0	0
g	0	0	0	0	0	0	0

(d)

	ac	ad	af	bd	be	cf	de	df
a	1	1	1	0	0	0	0	0
b	0	0	0	1	1	0	0	0
c	1	0	0	0	0	1	0	0
d	0	1	0	1	0	0	1	1
e	0	0	0	0	1	0	1	0
f	0	0	1	0	0	1	0	1
g	0	0	0	0	0	0	0	0

(e)

Graph Representation

- Which representation is best? It depends on the problem at hand.
- If the task is to process vertices adjacent to a vertex *v,* then the adjacency list requires only *deg*(*v*) steps, whereas the adjacency matrix requires $|V|$ steps.
- On the other hand, inserting or deleting a vertex adjacent to *v* requires linked list maintenance for an adjacency list (if such an implementation is used); for a matrix, it requires only changing 0 to 1 for insertion, or 1 to 0 for deletion, in one cell of the matrix.

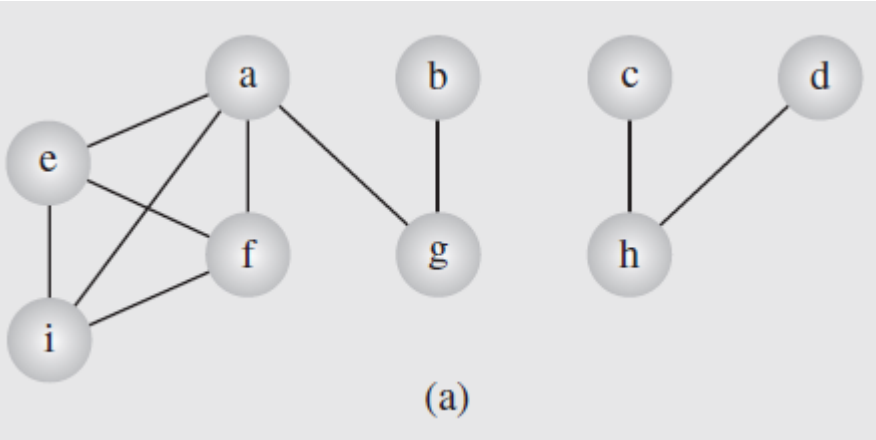
Graph Traversals

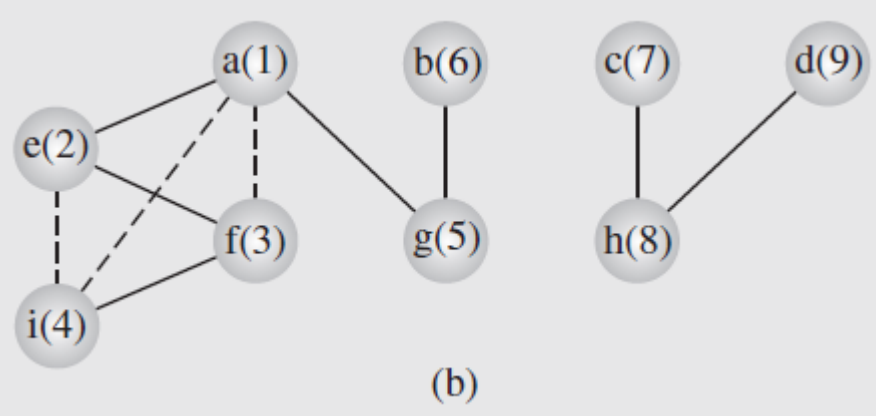
- As in trees, traversing a graph consists of visiting each vertex only one time.
- The simple traversal algorithms used for trees cannot be applied here because graphs may include cycles (can cause infinite loops).
- To prevent that from happening, each visited vertex can be marked to avoid revisiting it.
- However, graphs can have isolated vertices, which means that some parts of the graph are left out if unmodified tree traversal methods are applied.

Graph Traversals – Depth First Search

- In depth first search (traversal), each vertex *v* is visited and then each unvisited vertex adjacent to *v* is visited.
- If a vertex *v* has no adjacent vertices or all of its adjacent vertices have been visited, we backtrack to the predecessor of *v.*
- The traversal is finished if this visiting and backtracking process leads to the first vertex where the traversal started.
- If there are still some unvisited vertices in the graph, the traversal continues restarting for one of the unvisited vertices.

Graph Traversals – Depth First Search






```
DFS(v)
    num(v) = i++;
    for all vertices u adjacent to v
        if num(u) is 0
            attach edge(uv) to edges;
            DFS(u);

depthFirstSearch()
    for all vertices v
        num(v) = 0;
    edges = null;
    i = 1;
    while there is a vertex v such that num(v) is 0
        DFS(v);
    output edges;
```

Graph Traversals – Depth First Search

- Note that this algorithm guarantees generating a tree (or a forest, a set of trees) that includes or spans over all vertices of the original graph.
- A tree that meets this condition is called a *spanning tree.*
- Due to the condition “if *num*(u) *is* 0”, certain edges in the original graph do not appear in the resulting tree.
- The edges included in this tree are called *forward edges* (or *tree edges*), and the edges not included in this tree are called *back edges* and are shown as dashed lines.

Graph Traversals – Depth First Search

- The following example shows application of the same depthFirstSearch algorithm for a digraph.
- Notice that the original graph results in three spanning trees, although we started with only two isolated subgraphs.