Graphs

Lecture 16

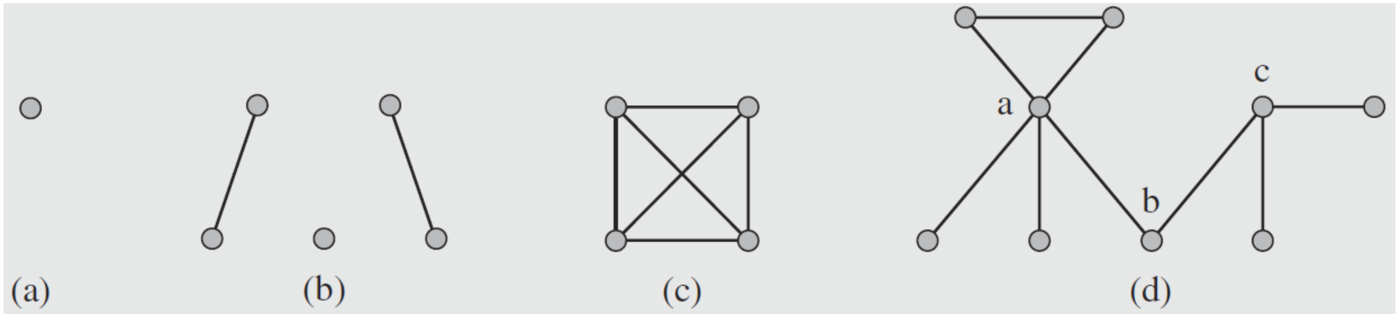
Instructor: Dr. Cong Pu, Ph.D.

cong.pu@okstate.edu

Adapted partially from Data Structures and Algorithms in Java, M.T. Goodrich, R.Tamassia and M. H. Goldwasser, Sixth Edition, Wiley; Data Structures and Algorithms in C++, Adam Drozdek, 4th Edition, Cengage Learning

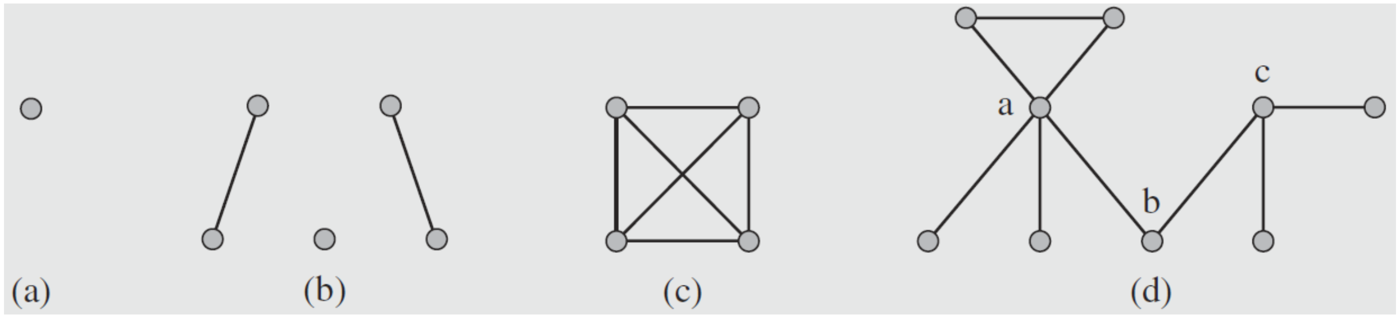
Introduction

- Trees
  - quite flexible, but inherent limitation -- only express *hierarchical* structures
- ***Graphs***
  - generalize a tree
  - a collection of nodes and the connections between them
    - no *restriction* on
      - # of vertices in the graph
      - # of connections one vertex can have to other vertices

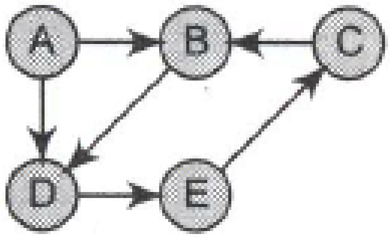


Terminologies

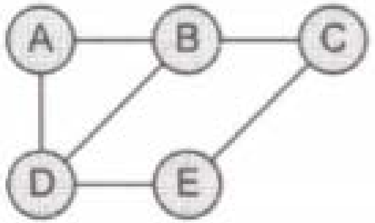
- A ***simple graph***
  - *G* = (*V*, *E*) consists of a *nonempty* set *V* of vertices and a *possibly empty* set *E* of edges, each edge being a set of two vertices from *V*
  - *V*, called a **vertex** or a ***point*** or a ***node***
  - *E*, *called* an **edge** or a ***line*** or a ***link***
  - # of vertices and edges denoted by |*V*| and |*E*|



Terminologies (cont.)

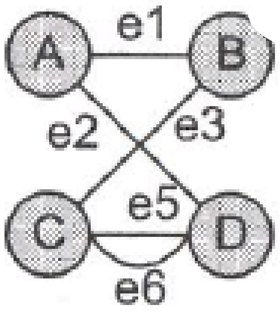


Directed graph

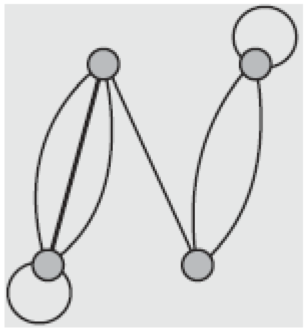


Undirected graph

- A ***directed graph, digraph***
  - G = (V, E), **(v_i, v_j) != (v_j, v_i)**
  - in a simple graph (undirected graph), (v_i, v_j) = (v_j, v_i)
- A **multigraph**
  - two vertices can be joined by **multiple edges**
- A **pseudograph**
  - a multigraph allowing for **loops**
  - a vertex can be joined with itself by an edge



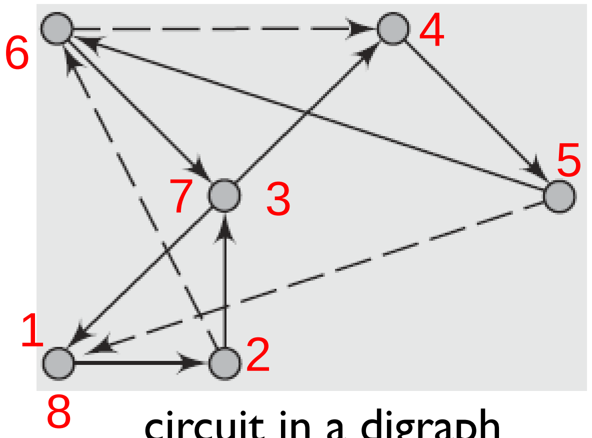
(a) Multi-graph



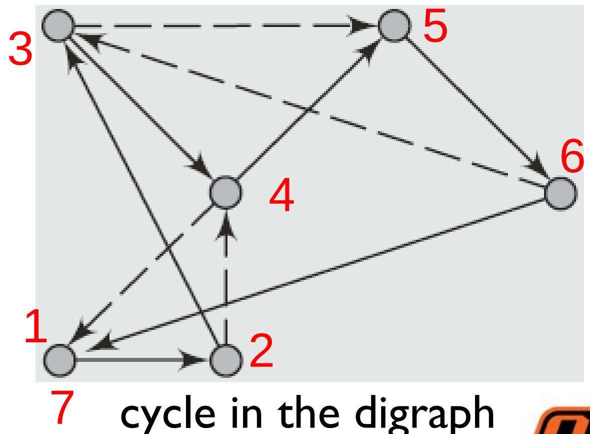
pseudograph

Terminologies (cont.)

- A ***path*** from v_1 to v_n
  - a sequence of edges, edge(v_1, v_2), edge(v_2, v_3), …, edge(v_{n-1}, v_n)
  - 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**,
    - **circuit**
  - if the vertices in a circuit are different,
    - **cycle**



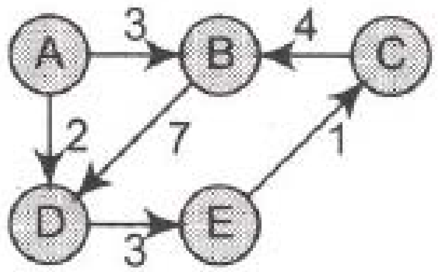
circuit in a digraph



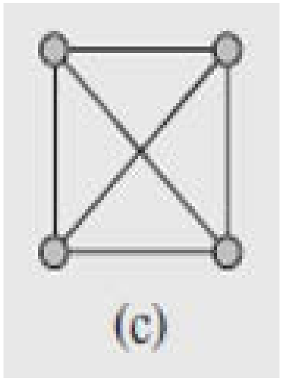
cycle in the digraph

Terminologies (cont.)

- A **weighted graph**
  - an assigned number (e.g., weight, cost, distance, length, etc.) on each edge
- A **complete graph**
  - **exactly one edge** between each pair of ***distinct*** vertices



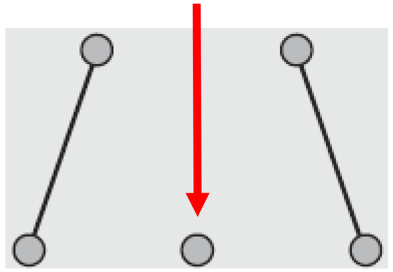
(c) Weighted graph



complete graph

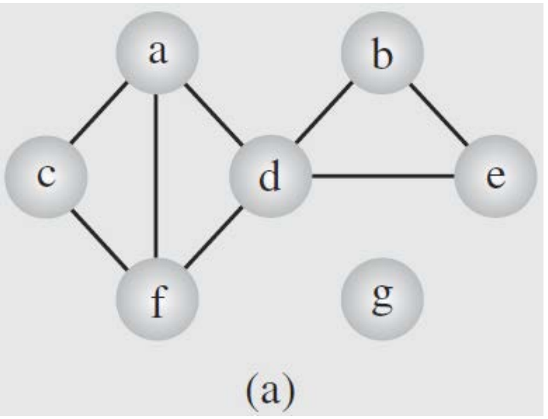
Terminologies (cont.)

- A ***subgraph G'*** of graph $G = (V, E)$,
  - $G' = (V', E')$, where V' € V and E' € E
- v_i and v_j are **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
- The **degree** of a vertex v,
  - deg(v), the **number of edges** incident with v
  - if deg(v) = 0, v is an **isolated vertex**

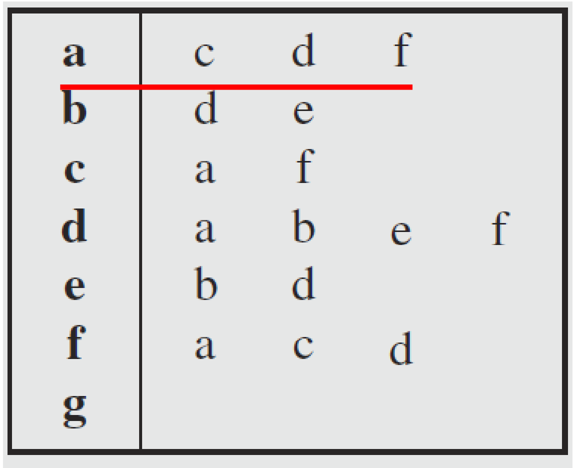


Graph Representation

- An ***adjacency list***
  - specify ***all vertices*** adjacent to ***each vertex*** of the graph

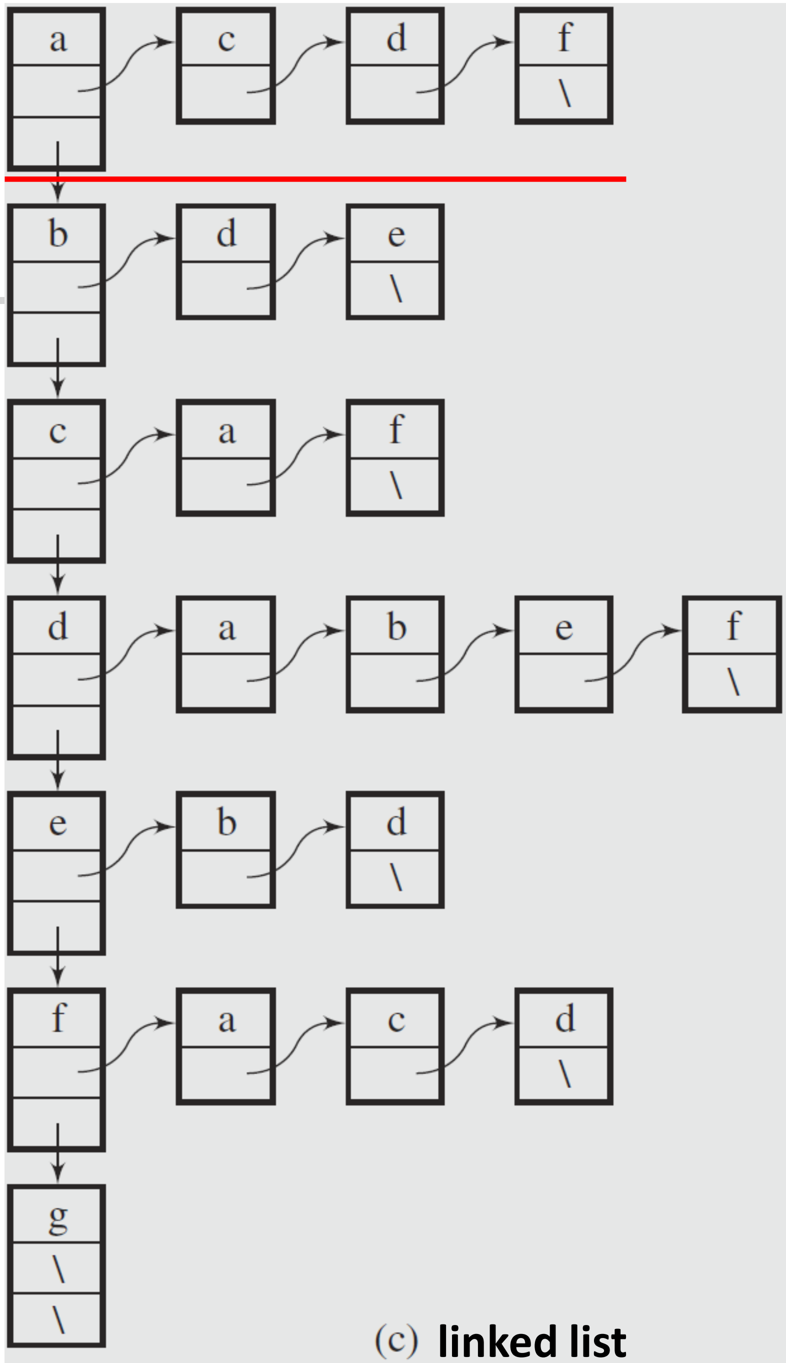


(a)



a	c d f
b	d e
c	a f
d	a b e f
e	b d
f	a c d
g	

(b) **table**

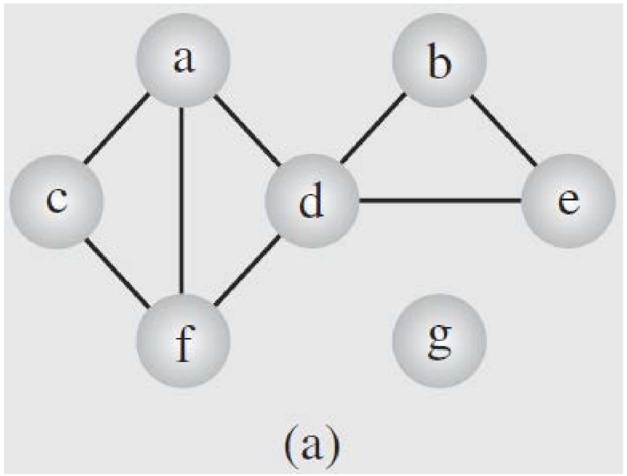


(c) **linked list**

Graph Representation (cont.)

- An ***adjacency matrix***
  - a $|V| \times |V|$ ***binary matrix*** where each entry a_{ij} of the matrix

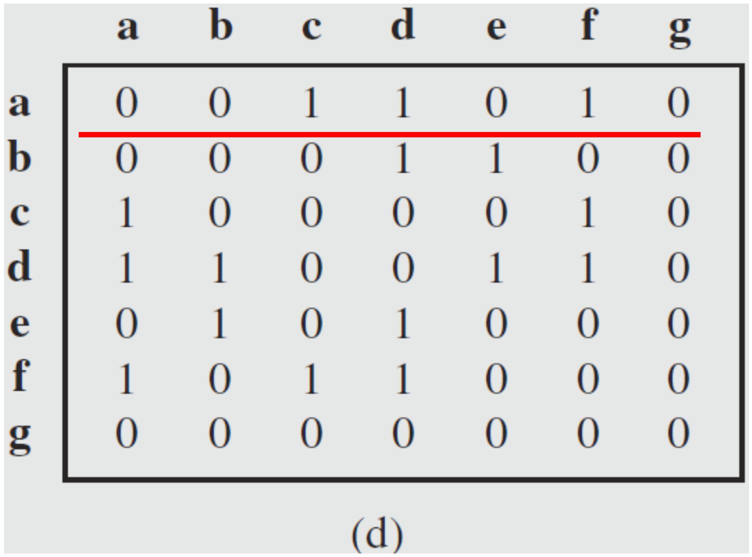
$$a_{ij} = \begin{cases} 1 & \text{if there exists an edge } (v_i v_j) \\ 0 & \text{otherwise} \end{cases}$$



(a)

vertices (arbitrary order)

vertices



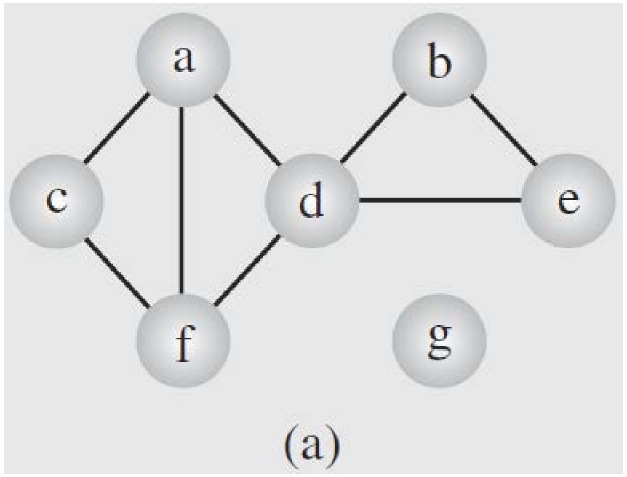
	a	b	c	d	e	f	g
a	0	0	1	1	0	1	0
b	0	0	0	1	1	0	0
c	1	0	0	0	0	1	0
d	1	1	0	0	1	1	0
e	0	1	0	1	0	0	0
f	1	0	1	1	0	0	0
g	0	0	0	0	0	0	0

(d)

Graph Representation (cont.)

- An ***incidence matrix***
  - a |*V*| x |E| ***binary matrix*** where each entry a_{ij} of the matrix

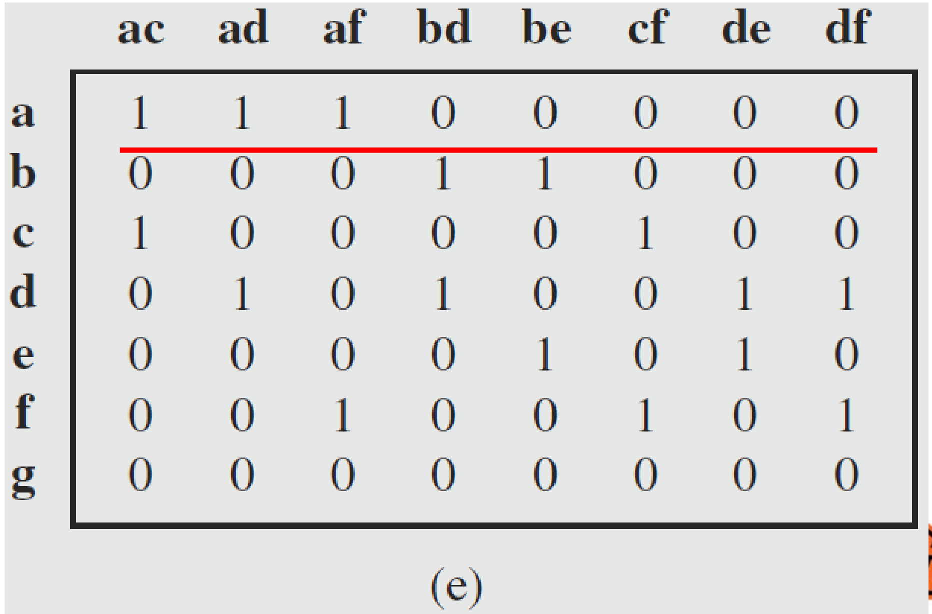
$$a_{ij} = \begin{cases} 1 & \text{if edge } e_j \text{ is incident with vertex } v_i \\ 0 & \text{otherwise} \end{cases}$$



(a)

edges

vertices

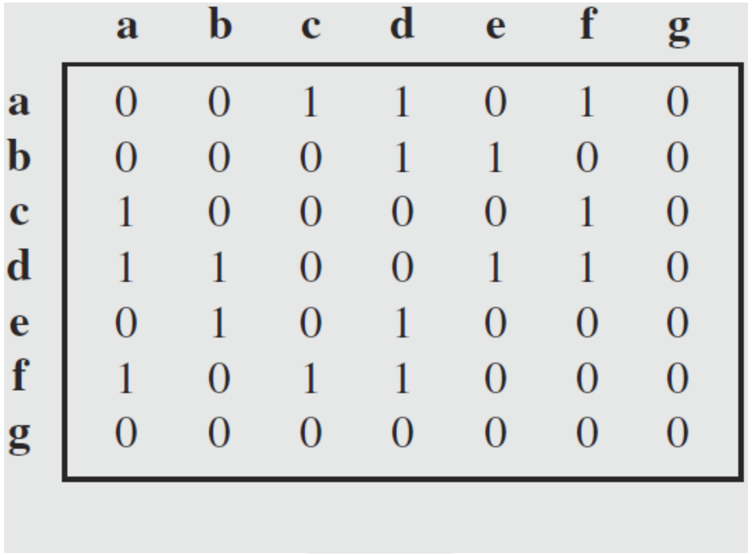


$$\begin{array}{c|cccccccc} & \mathbf{ac} & \mathbf{ad} & \mathbf{af} & \mathbf{bd} & \mathbf{be} & \mathbf{cf} & \mathbf{de} & \mathbf{df} \\ \hline \mathbf{a} & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{b} & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \mathbf{c} & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \mathbf{d} & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ \mathbf{e} & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ \mathbf{f} & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ \mathbf{g} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

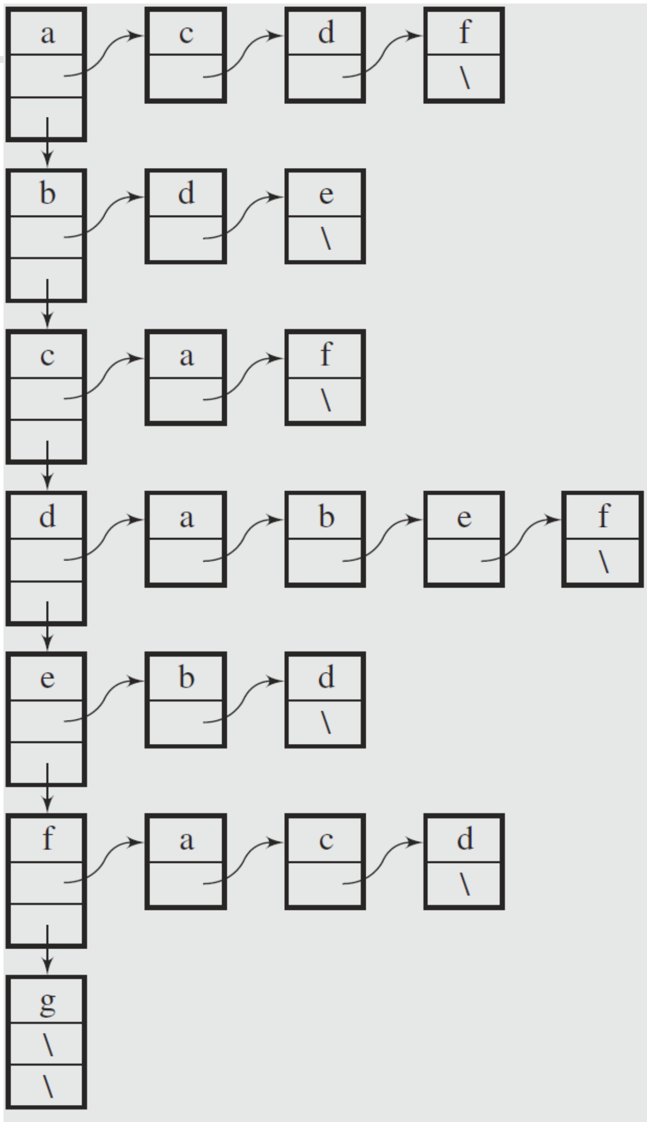
(e)

Graph Representation (cont.)

- Which **representation is best?** *it depends*
  - if process vertices adjacent to a vertex v,
    - adjacency list is better
  - if insert or delete a vertex adjacent to v,
    - matrix is better



	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



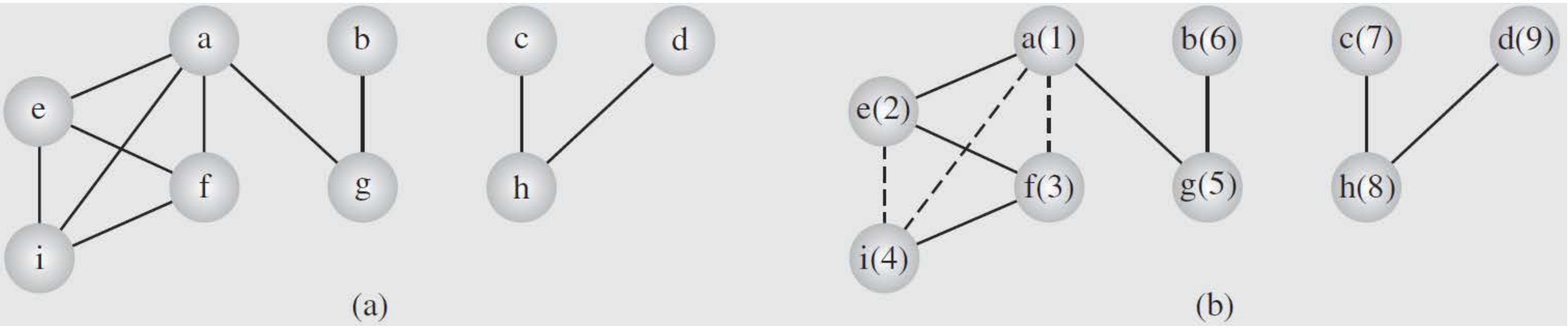
Graph Traversals

- ***Traversing a graph***: visit each node once
  - e.g., like tree traversals
  - cannot apply tree traversal algorithms to graphs because of ***cycles*** and ***isolated vertices***

- **Depth-first search**,
  - each vertex v is visited
  - all the <u>unvisited vertices</u> **adjacent** to vertex v are visited
  - if v has no adjacent vertices, or all of v's adjacent vertices already visited,
    - **backtrack** to v's <u>predecessor</u>
  - continue until we return to the vertex where the traversal started

Graph Traversals (cont.)

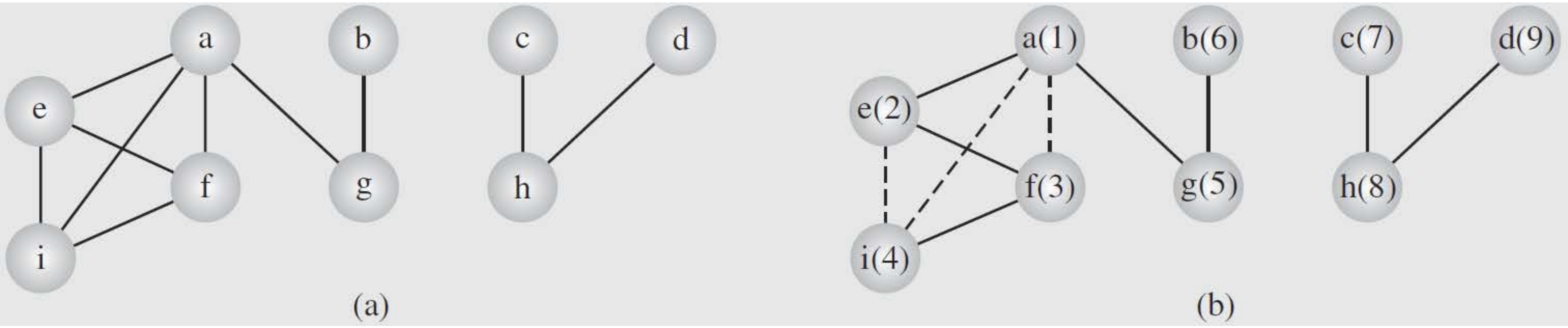
- **Depth-first search** (cont.),
  - if any vertices remain unvisited at this point,
  - restart the traversal at <u>one of the unvisited vertices</u>
  - e.g.,



Note: the numbers indicate the order in which the nodes are visited; the solid lines indicate the edges traversed during the search

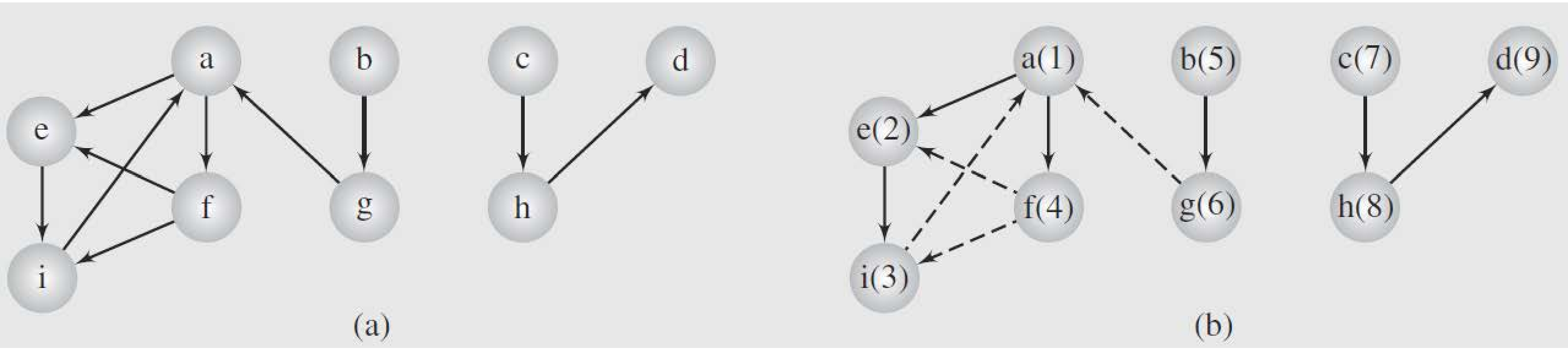
Graph Traversals (cont.)

- **Depth-first search** (cont.),
  - create a tree (or a forest, which is a set of trees) including the graph's vertices, called a **spanning tree**
  - the edges included in the tree are called **forward edges**; those omitted are called **back edges**



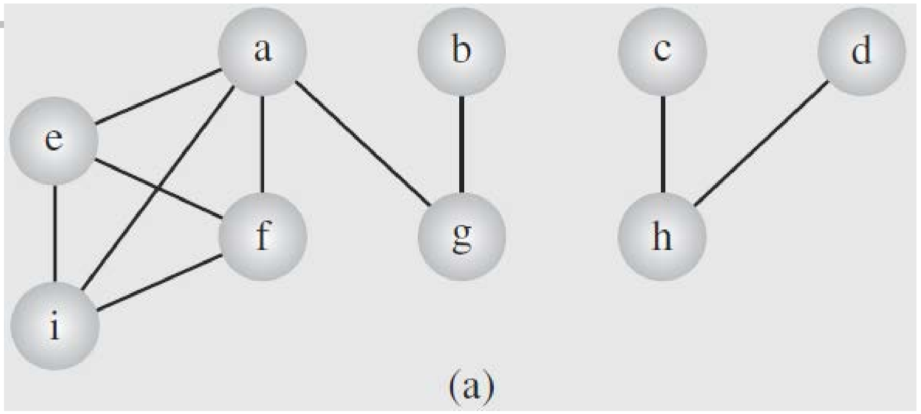
Graph Traversals (cont.)

- **Depth-first search** (cont.),
  - a directed graph case
  - a forest of **three trees**, because the traversal must follow the direction of the edges
  - more number of algorithms based on depth-first searching



Graph Traversals (cont.)

- **Depth-first search** (cont.),

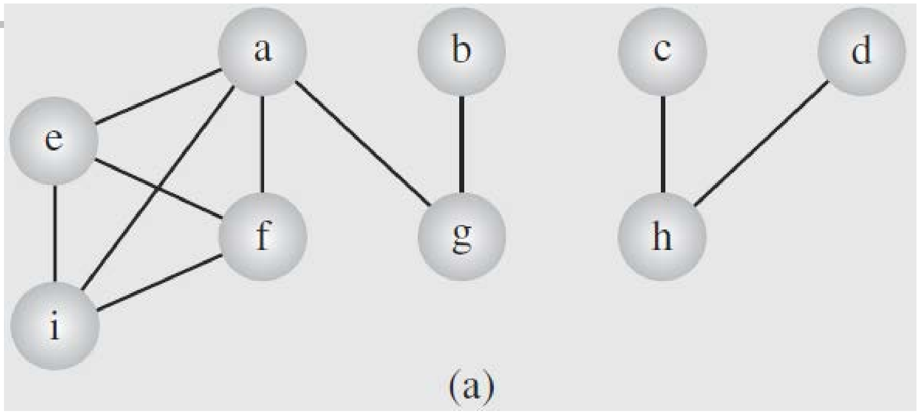


```
DFS(v)
    num(v)= i++;
    for all vertices u adjacent to v
        if num(u) is 0
            attach edge(uv) to edges;
            DFS(u);

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;
```

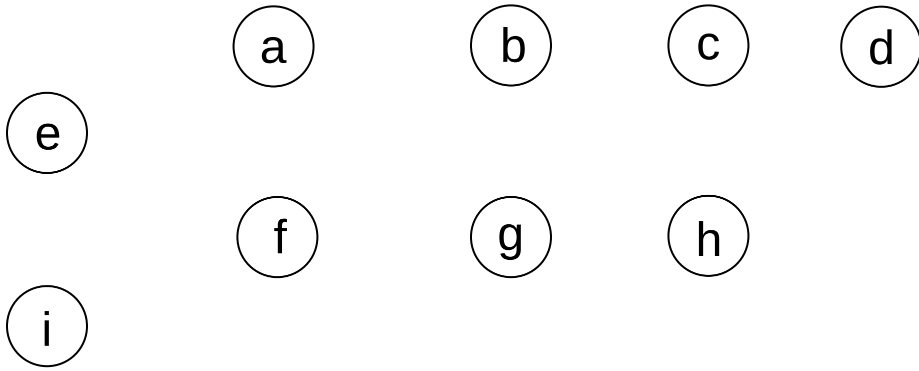
Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```



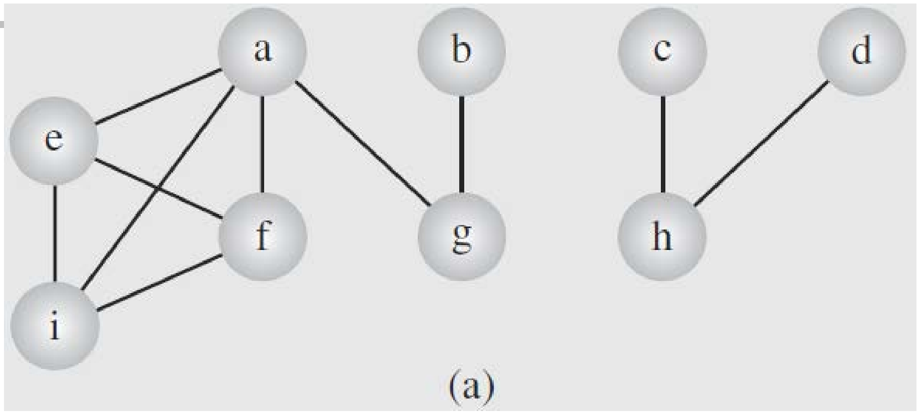
(a)

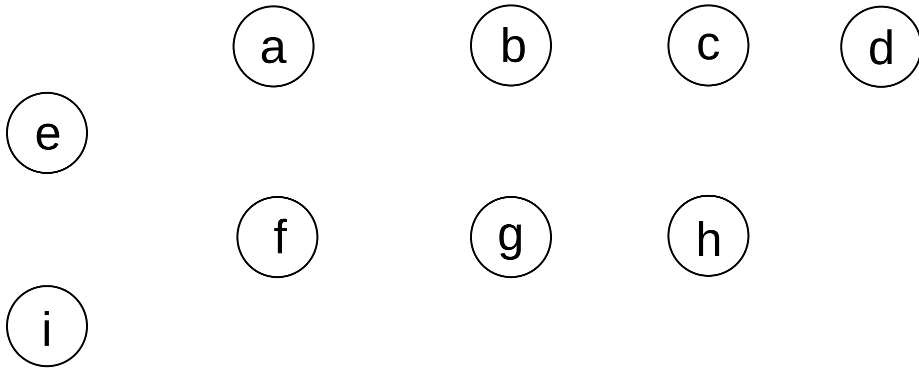


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

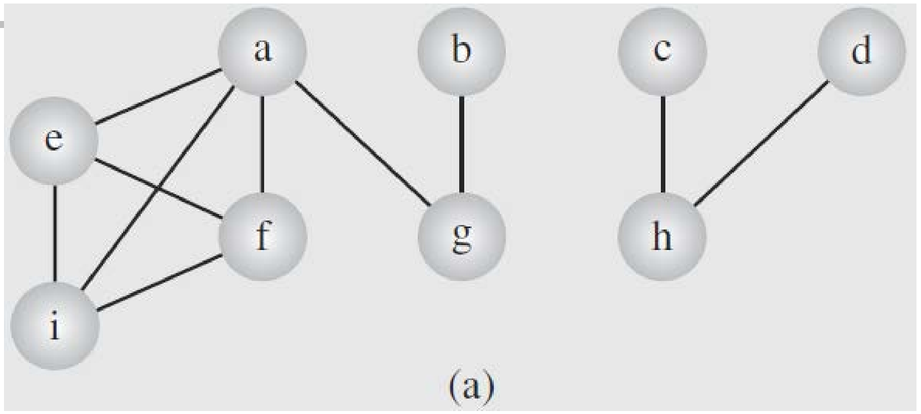


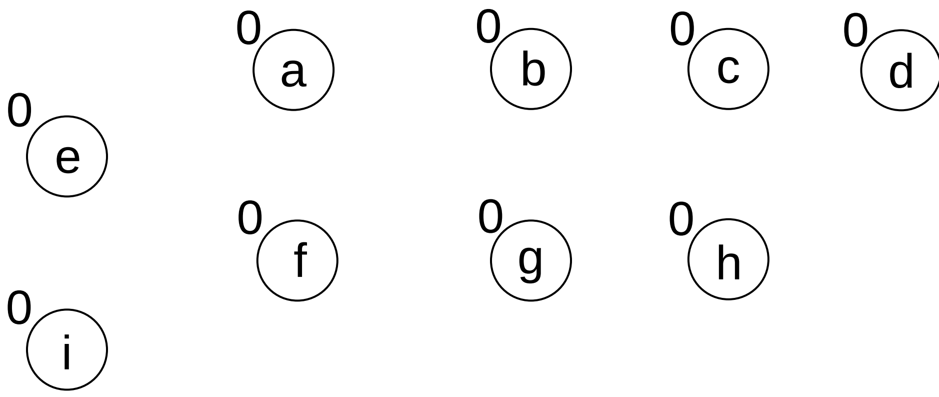


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

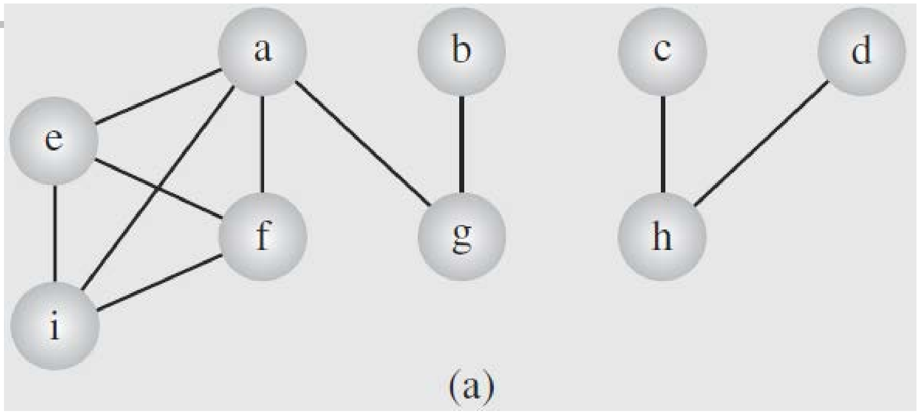




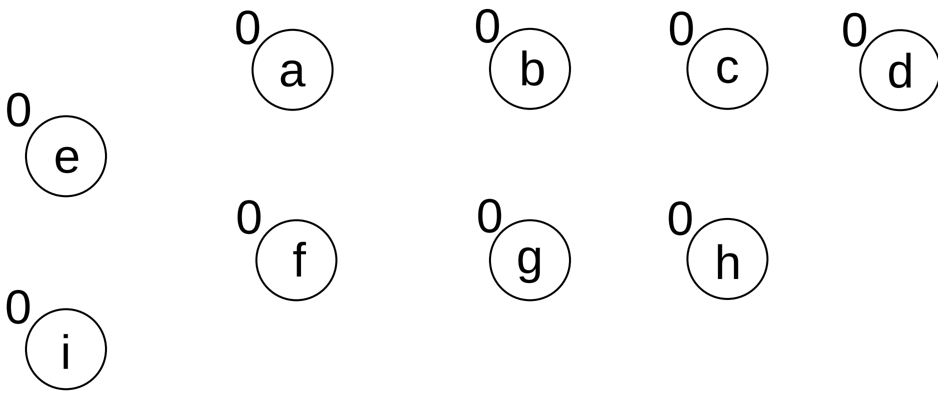
Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```



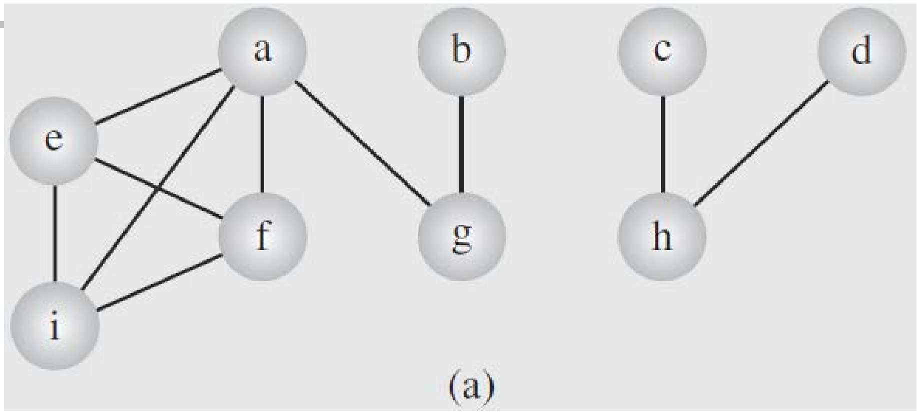
(a)

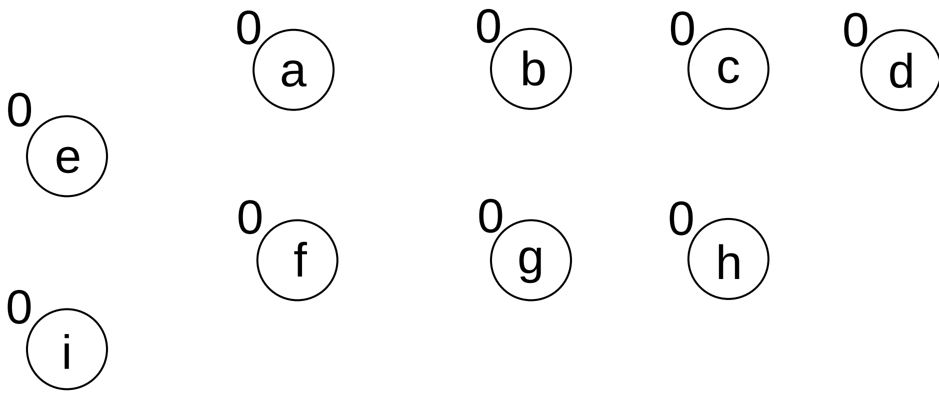


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

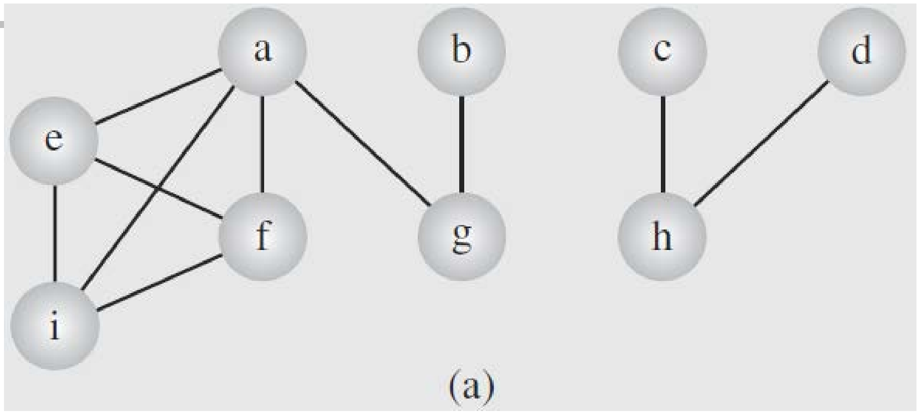




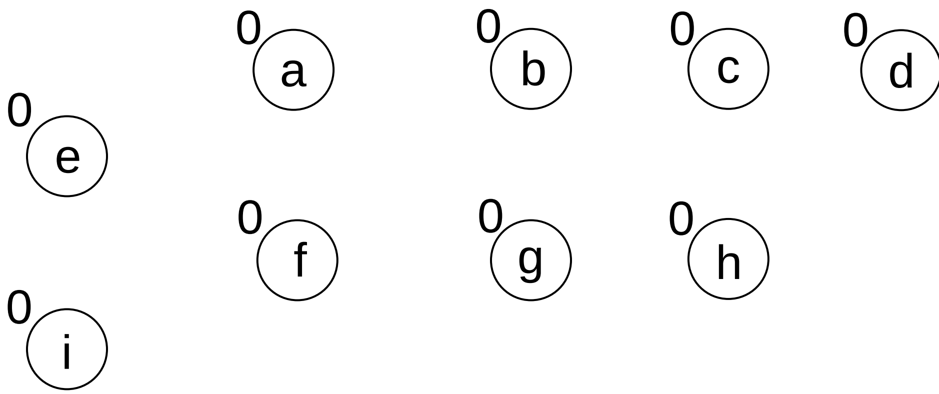
Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```



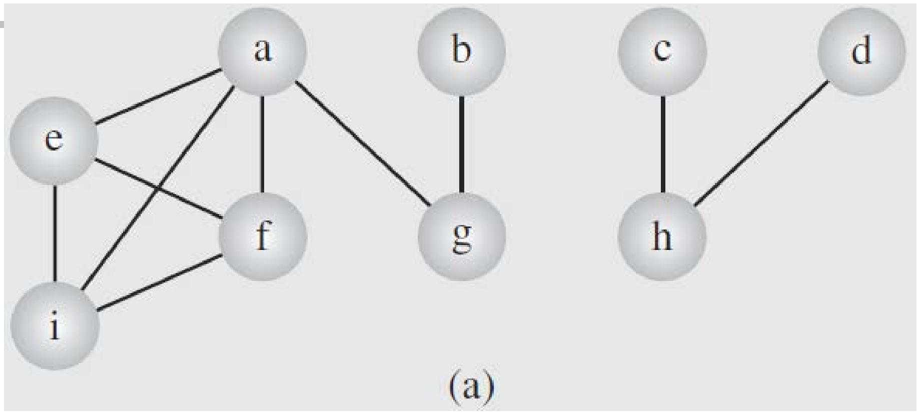
(a)



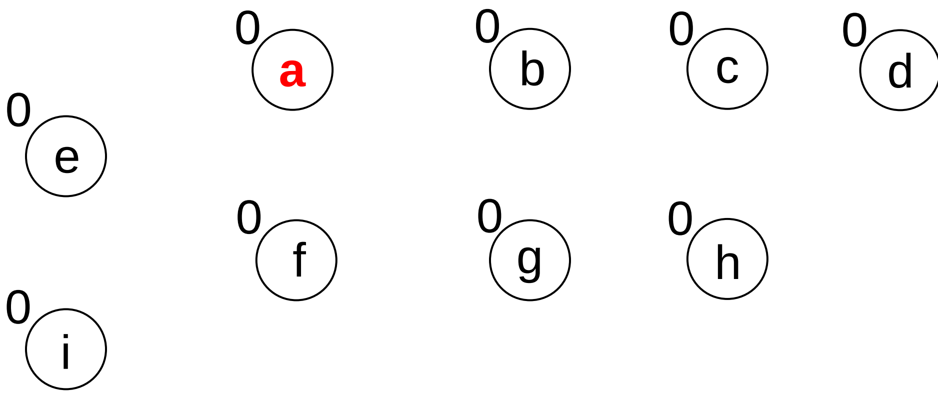
Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```



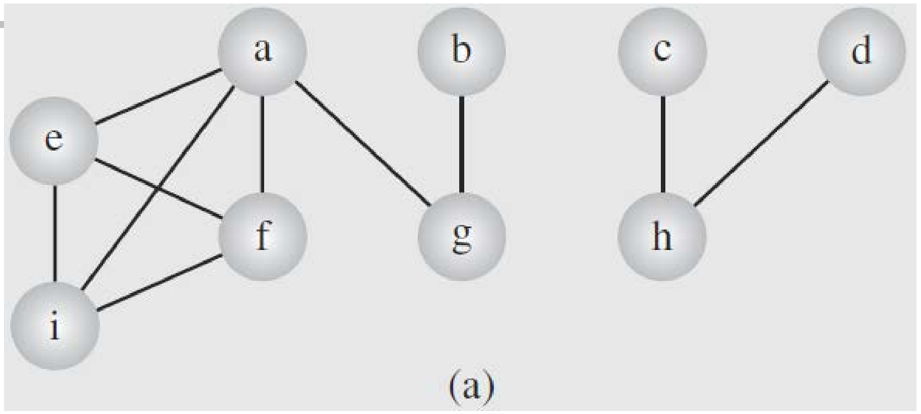
(a)



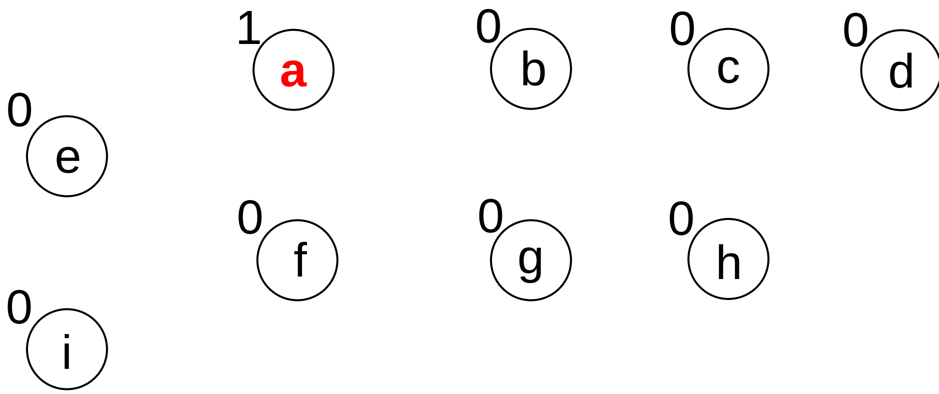
Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```



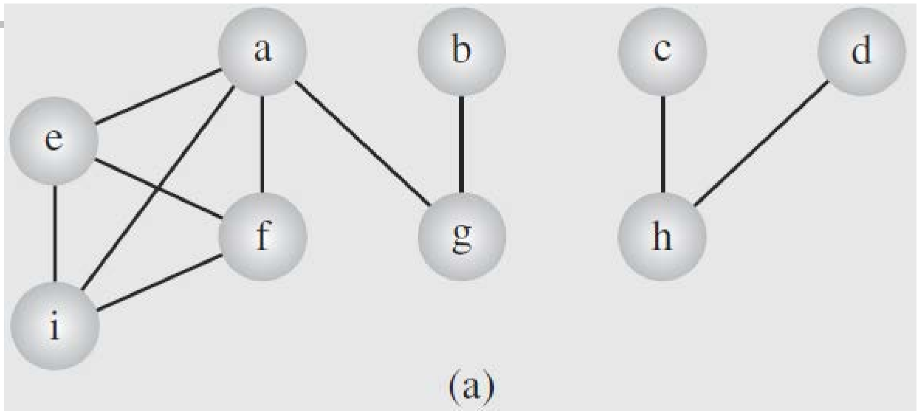
(a)

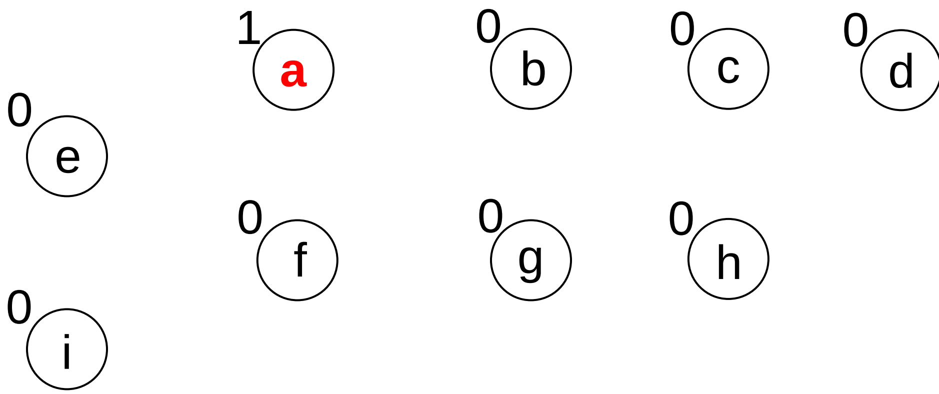


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

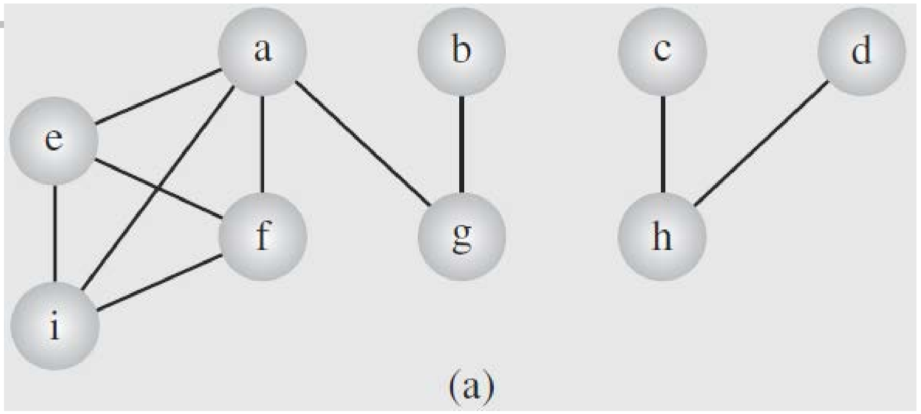


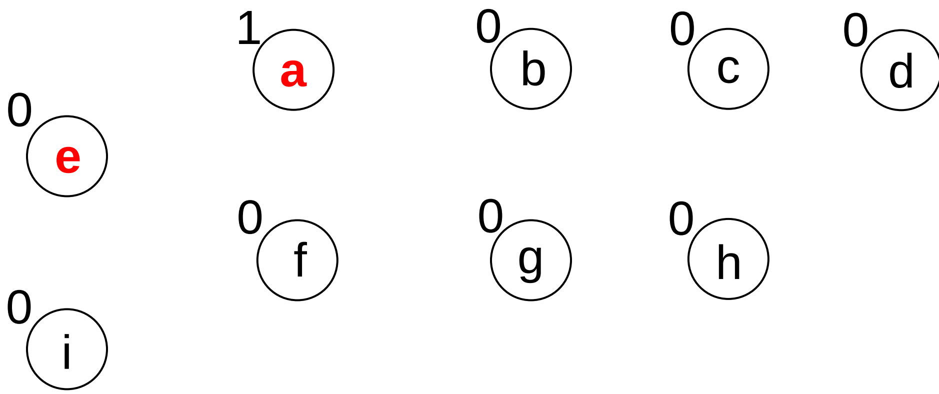


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

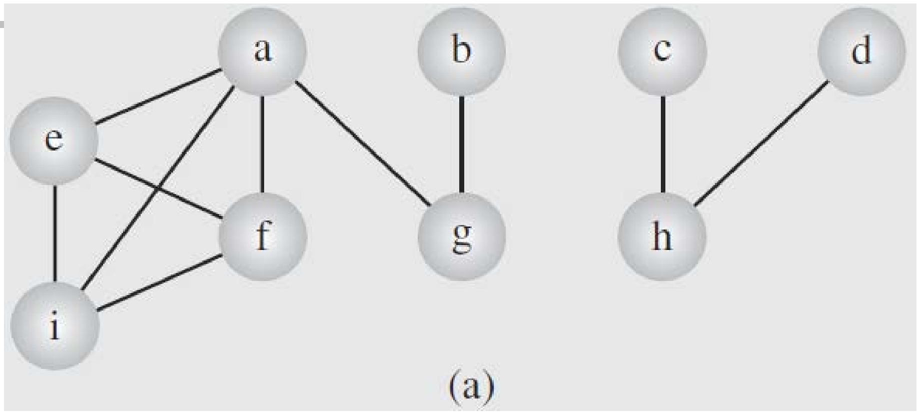


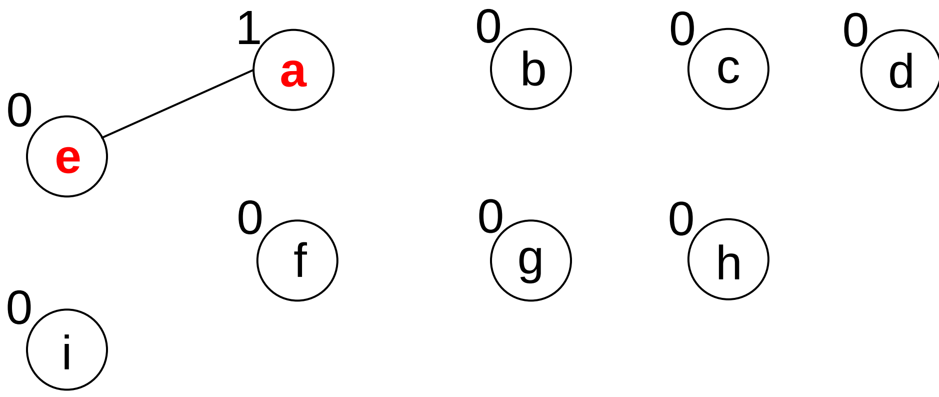


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

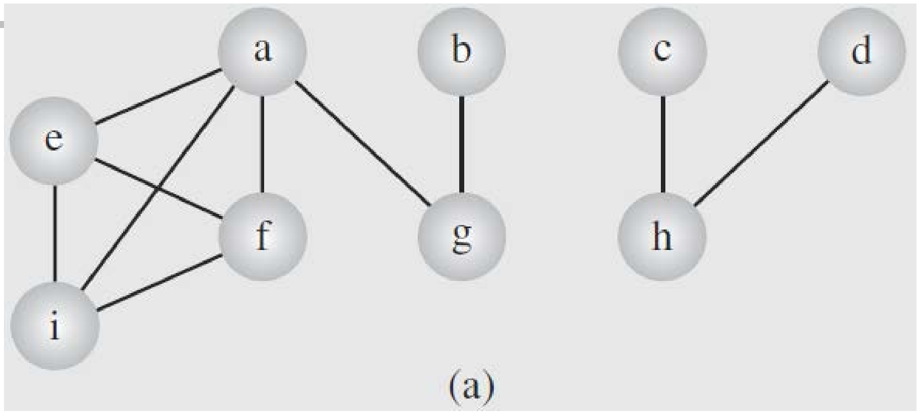


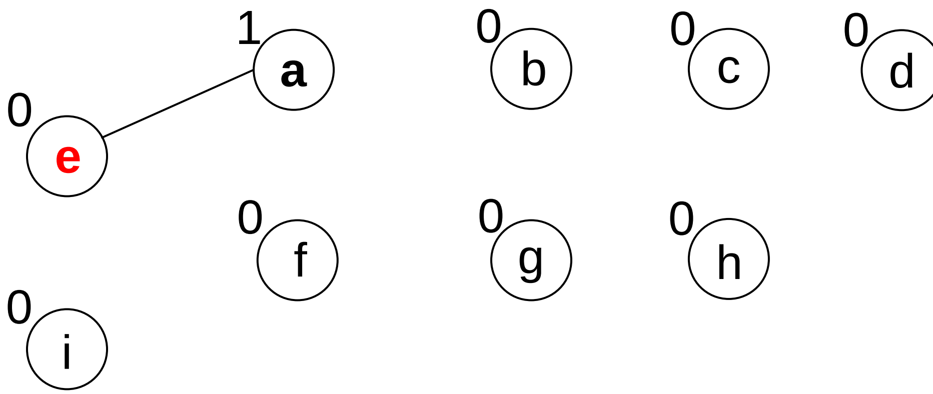


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

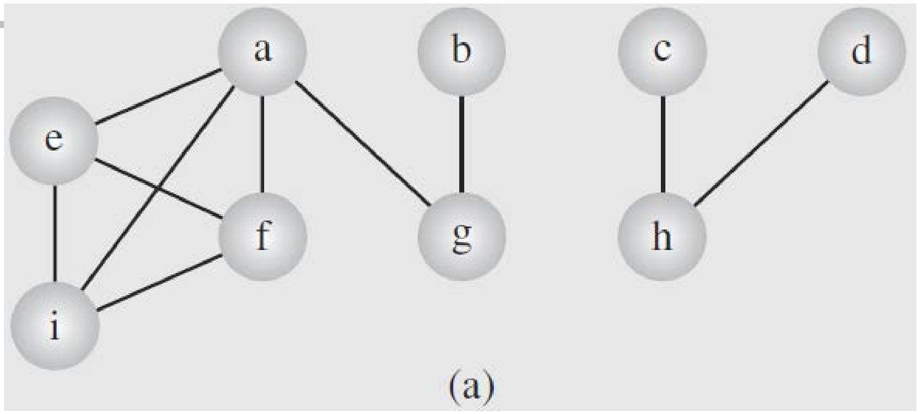




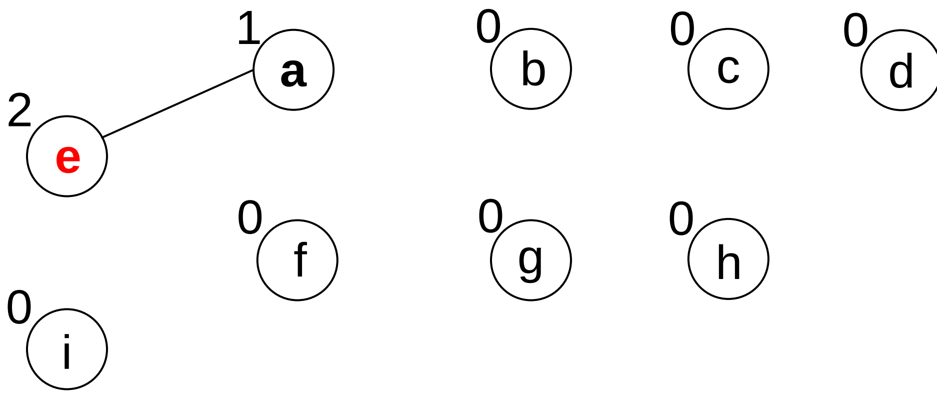
Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```



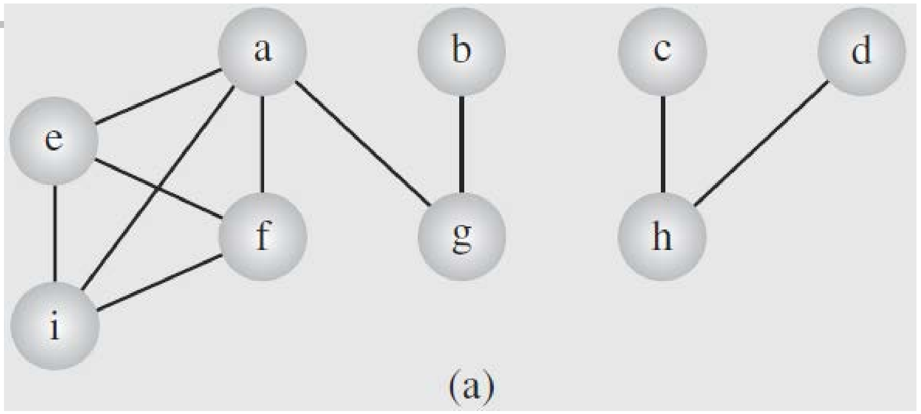
(a)

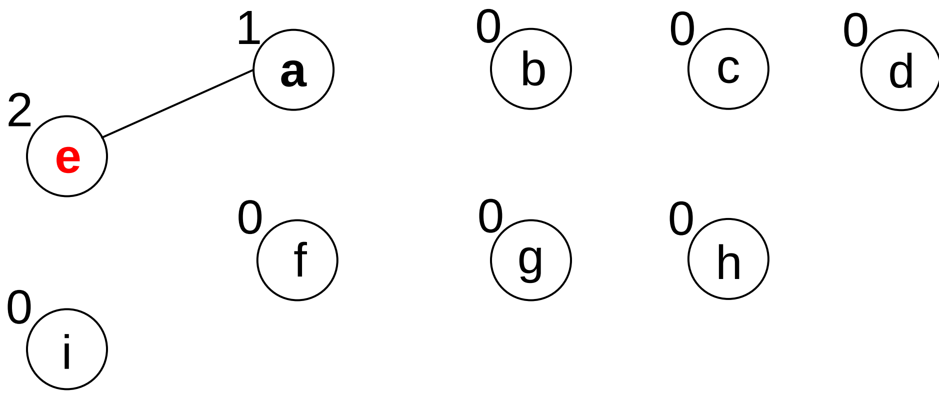


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

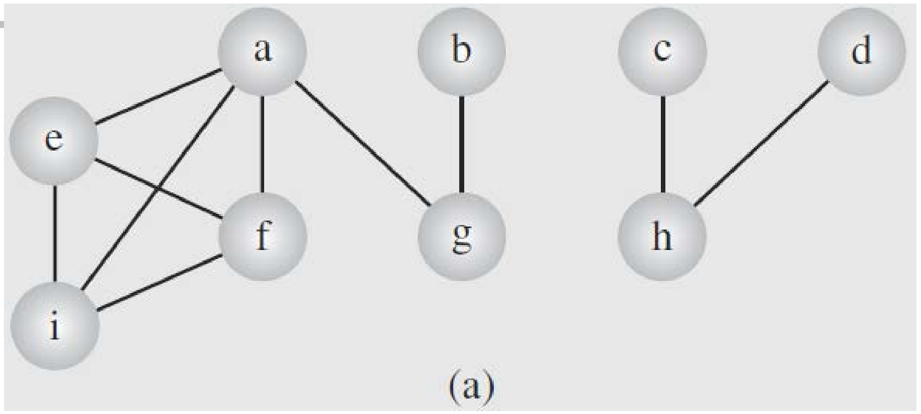


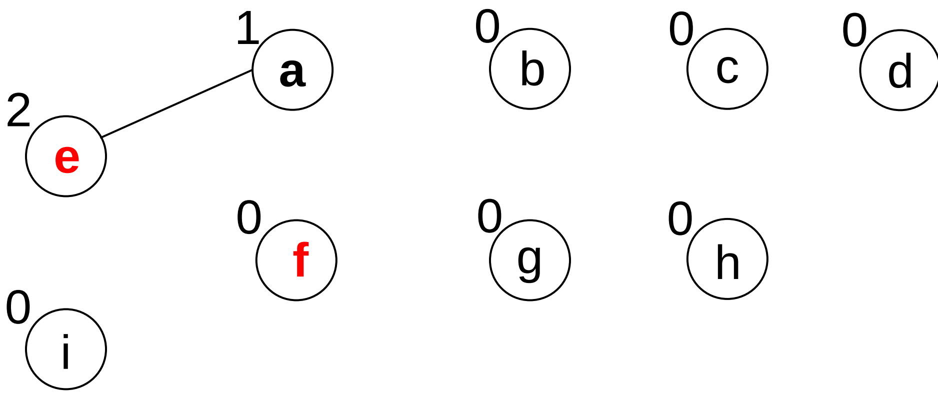


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

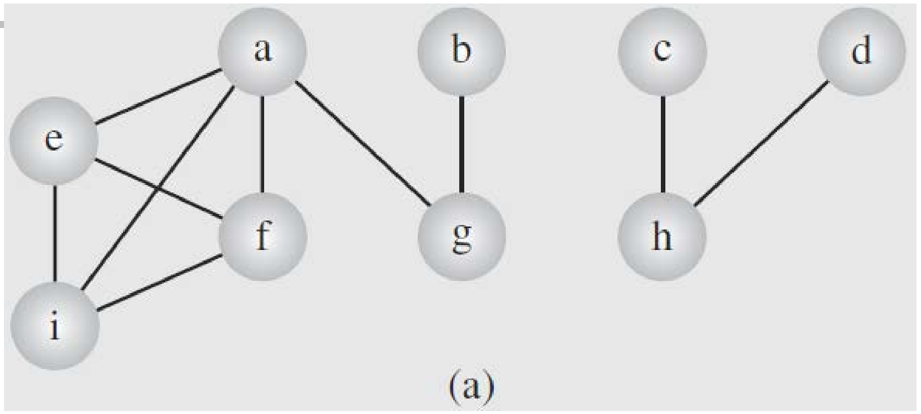


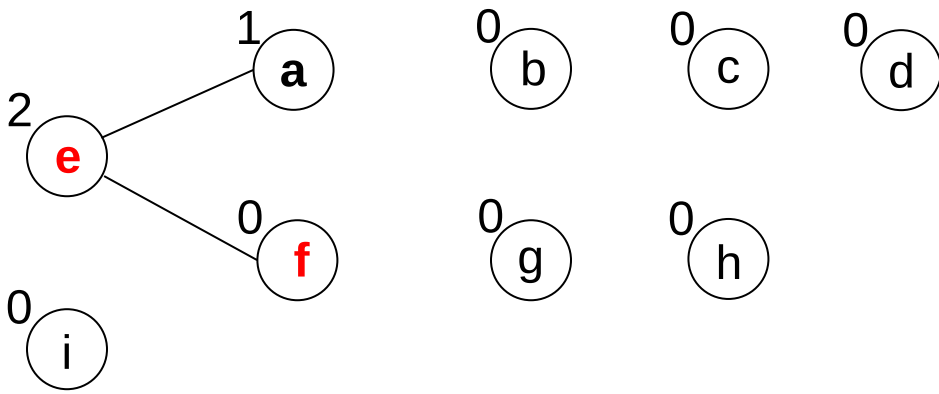


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

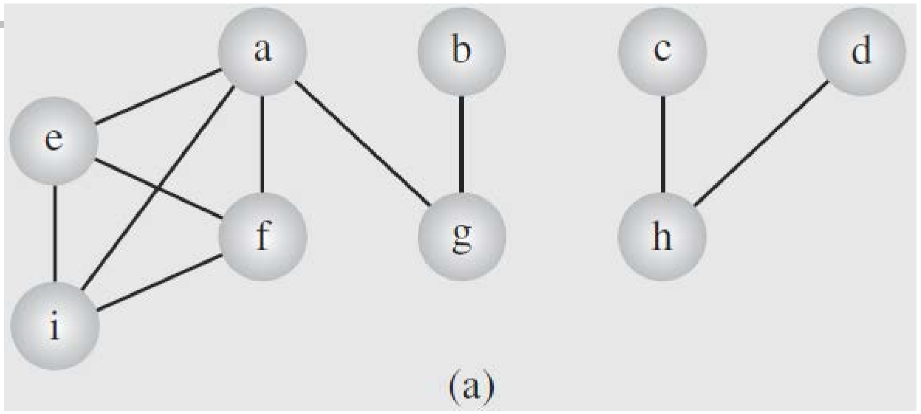




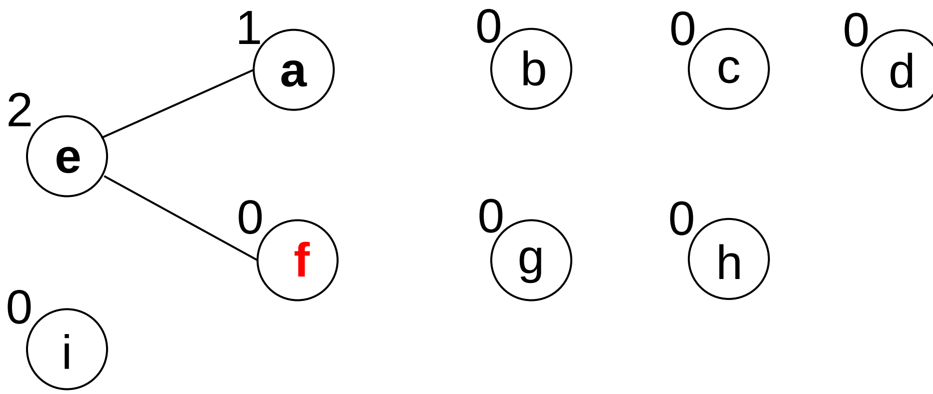
Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```



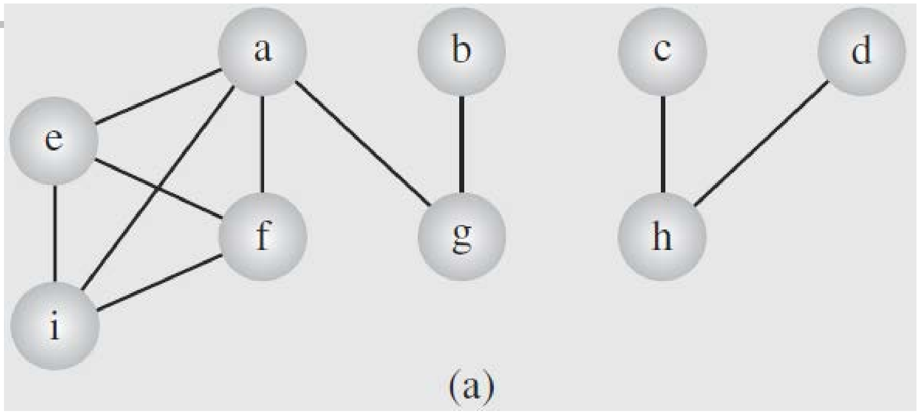
(a)



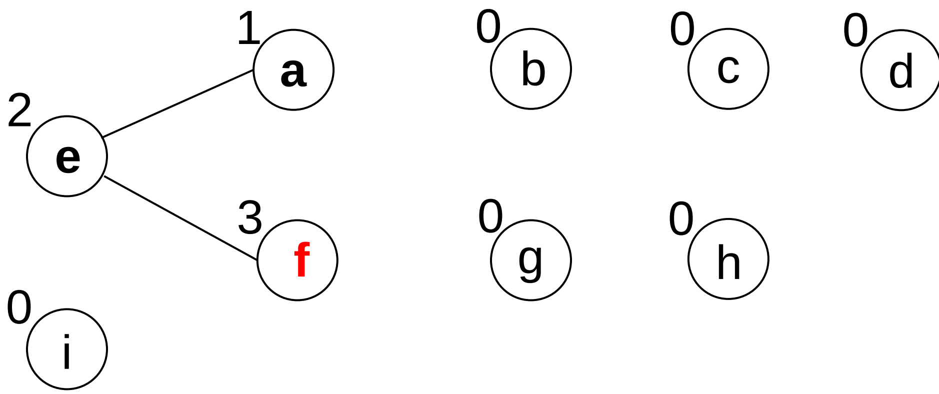
Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```



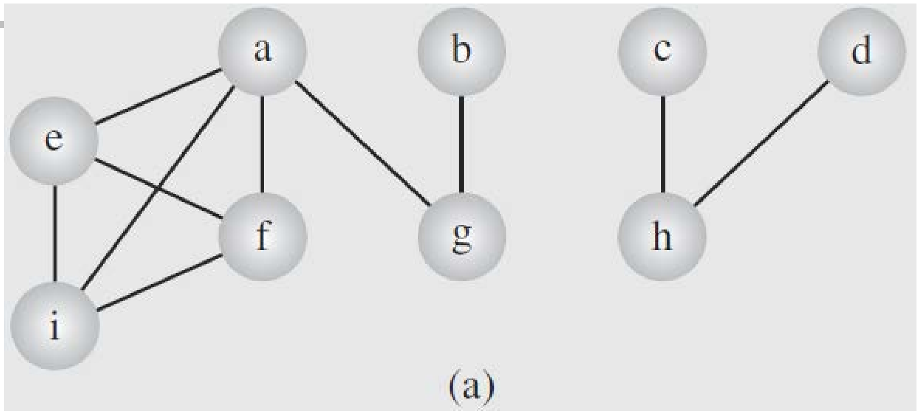
(a)

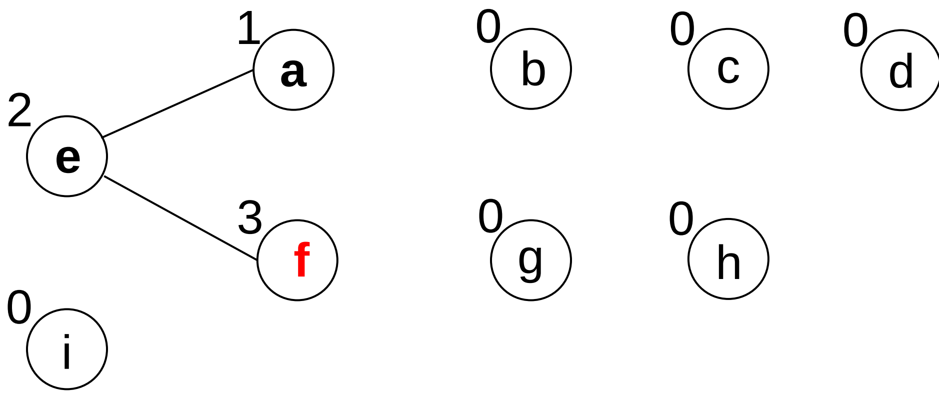


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

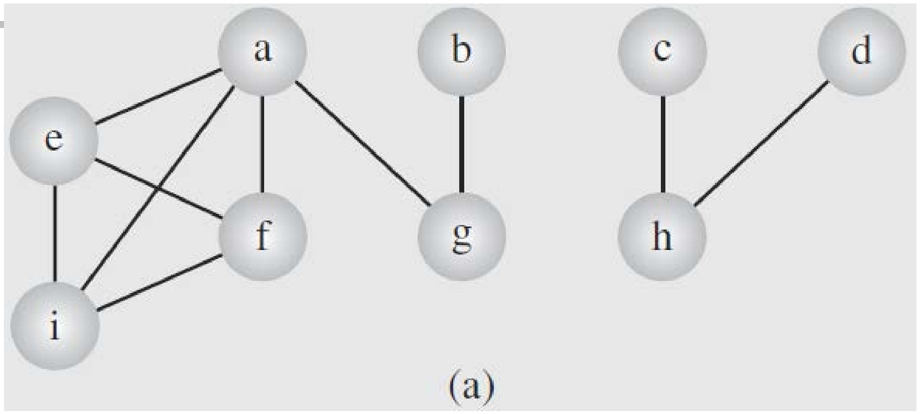


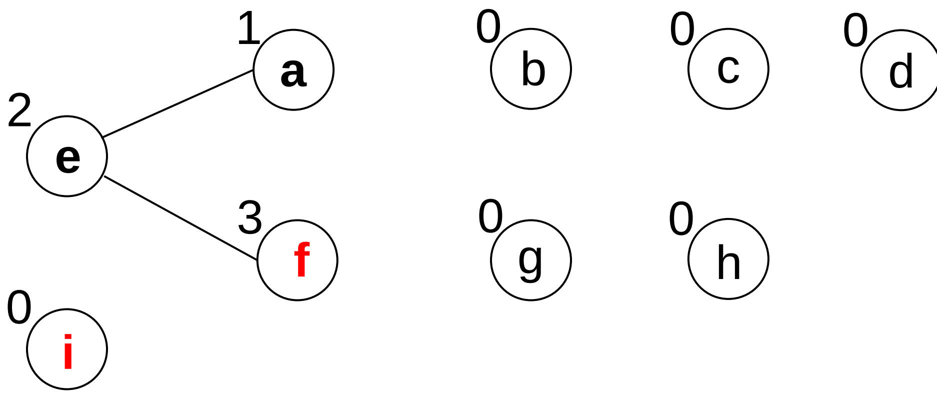


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

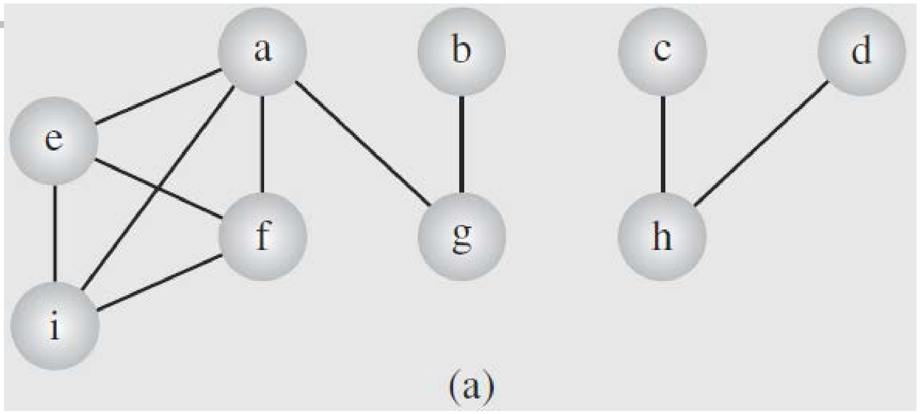


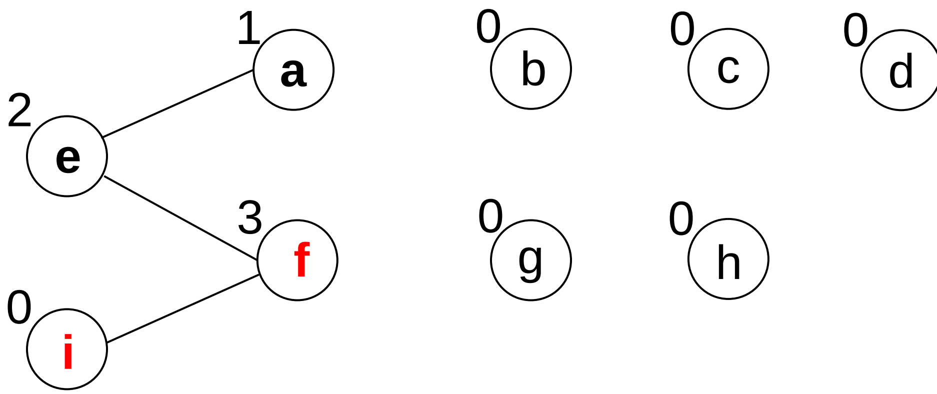


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

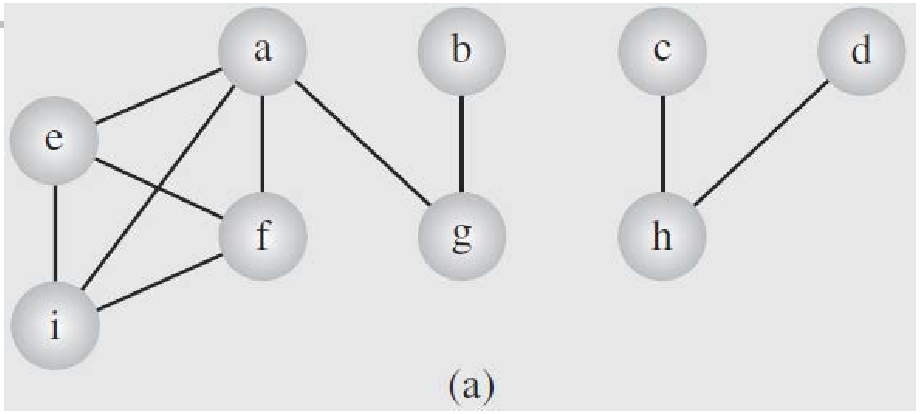


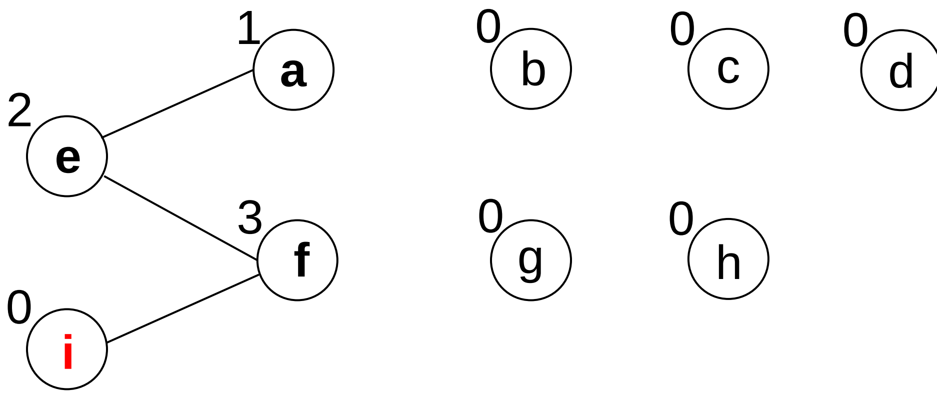


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

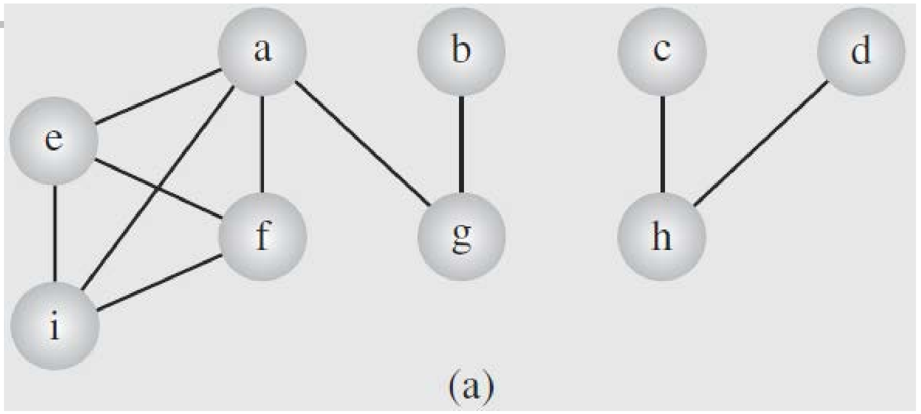


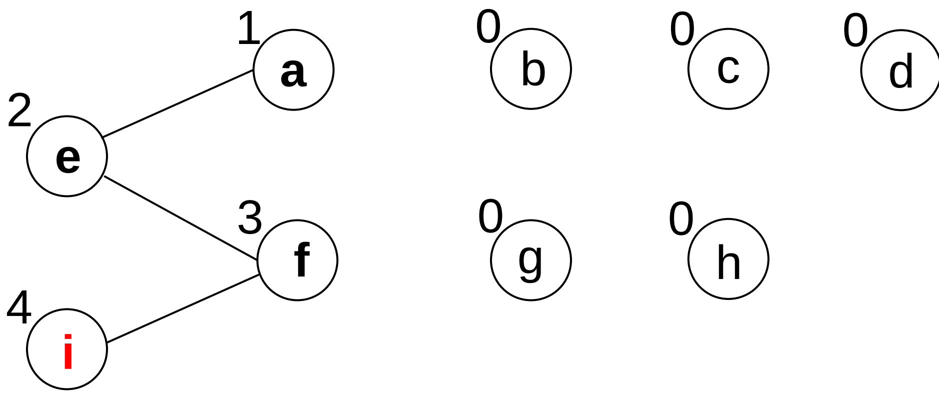


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

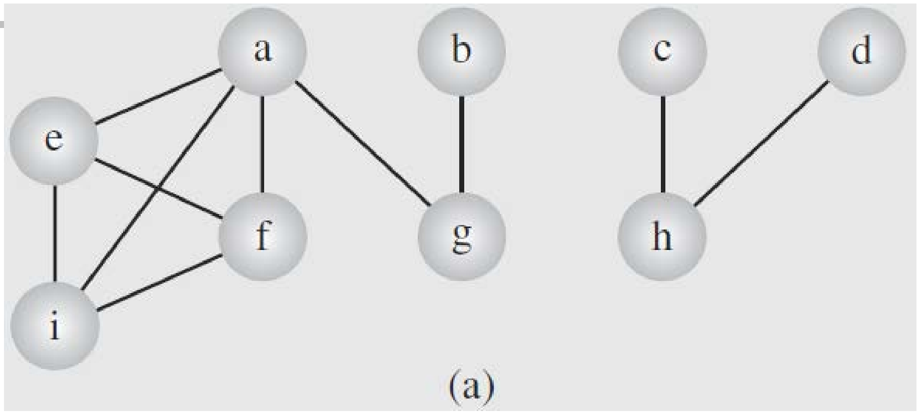


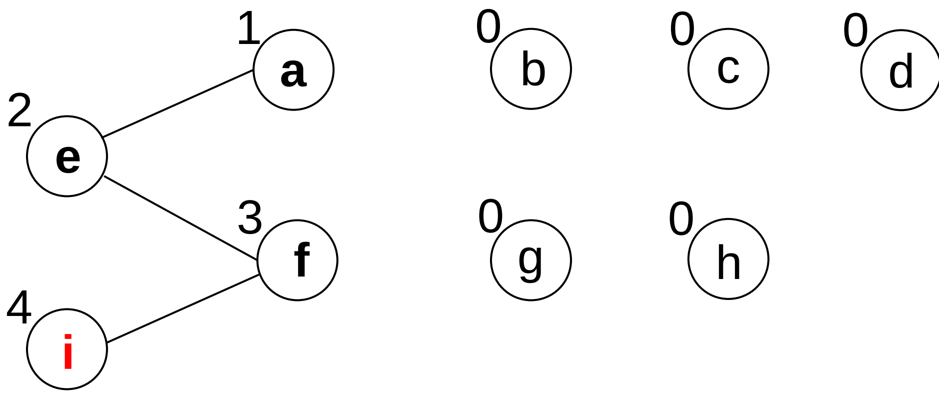


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

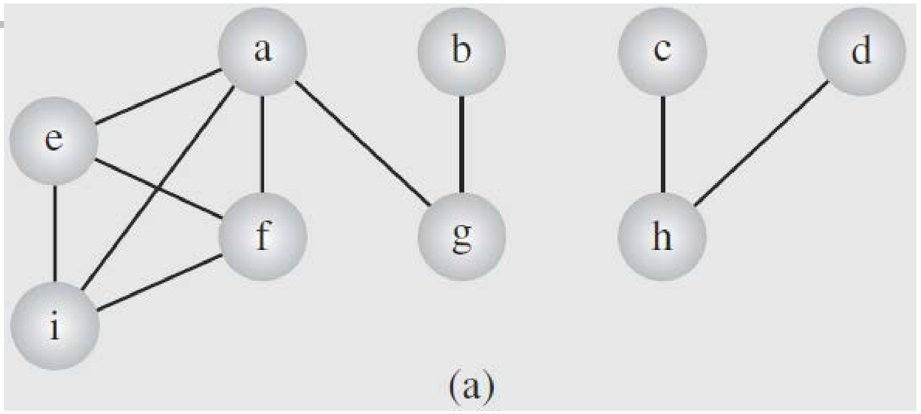




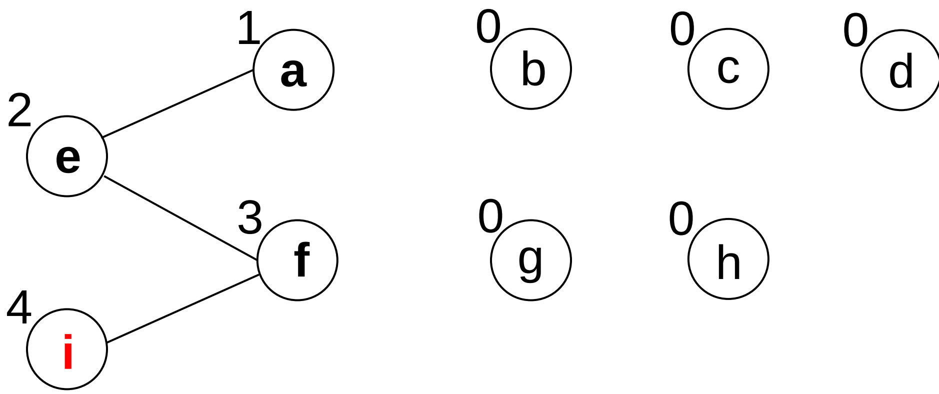
Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```



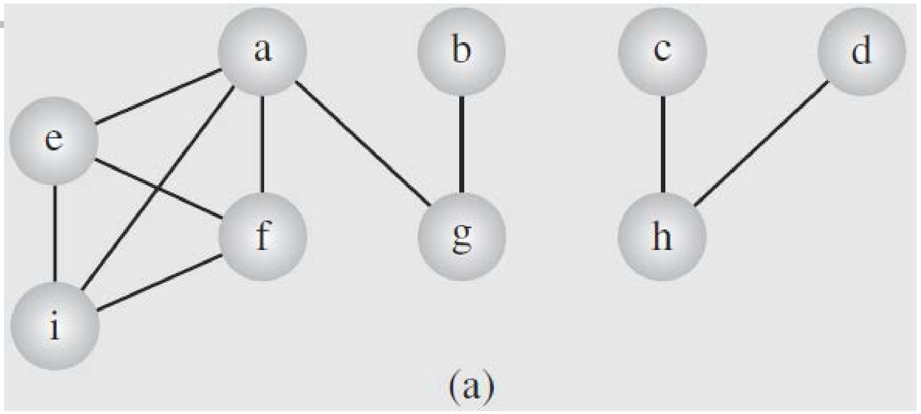
(a)



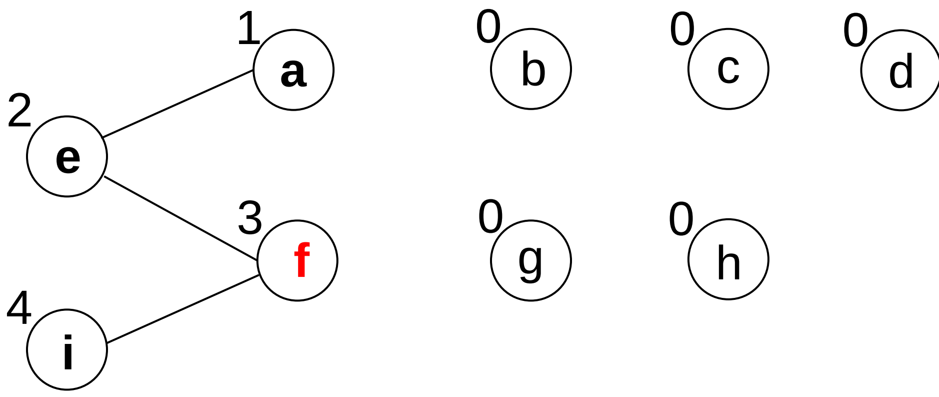
Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```



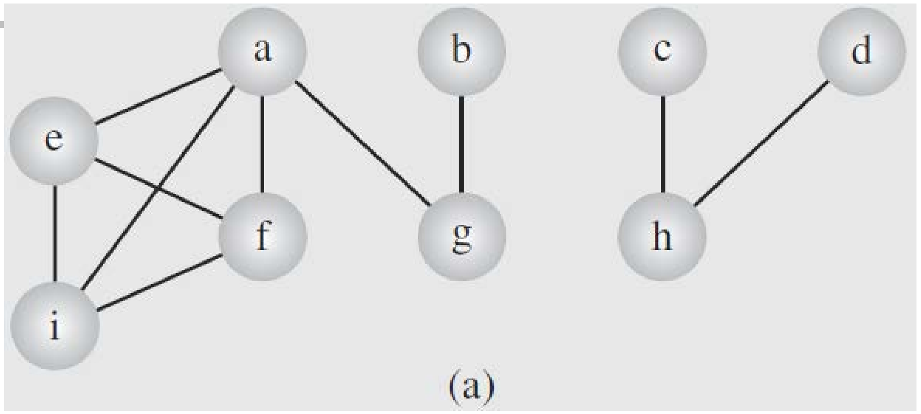
(a)

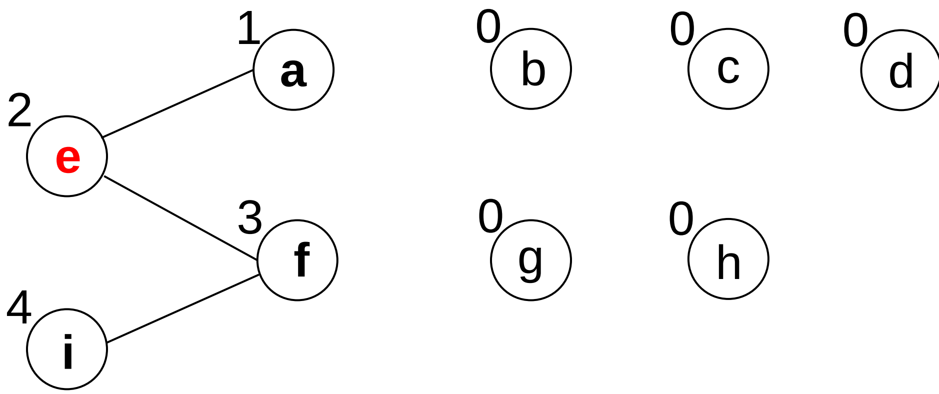


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

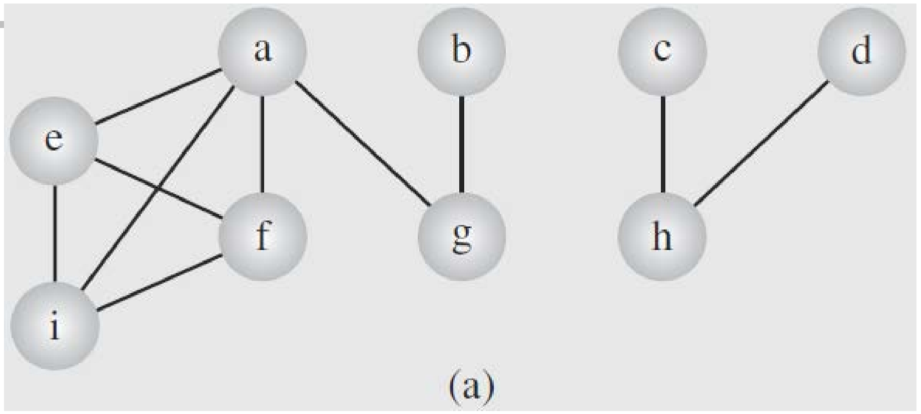




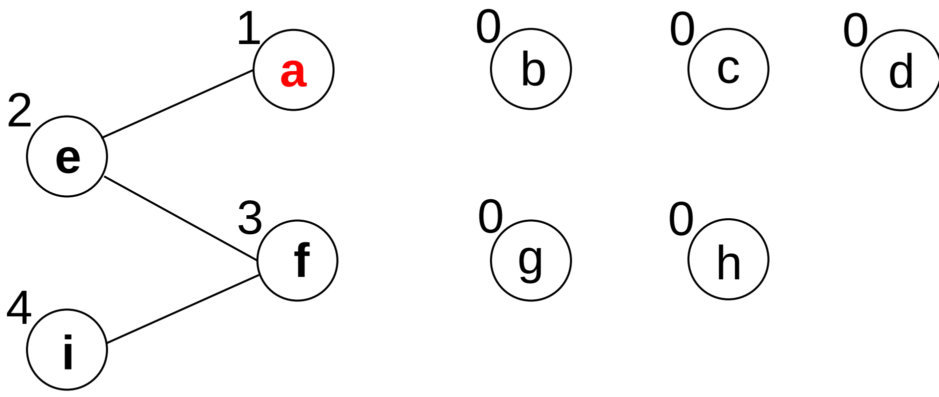
Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```



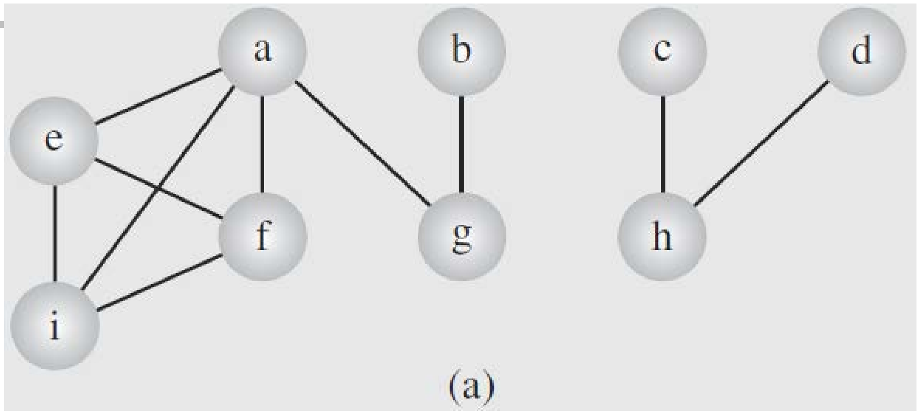
(a)

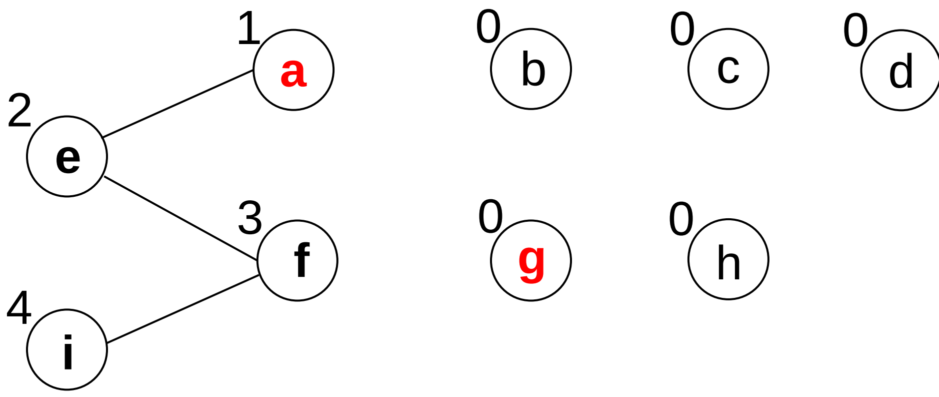


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

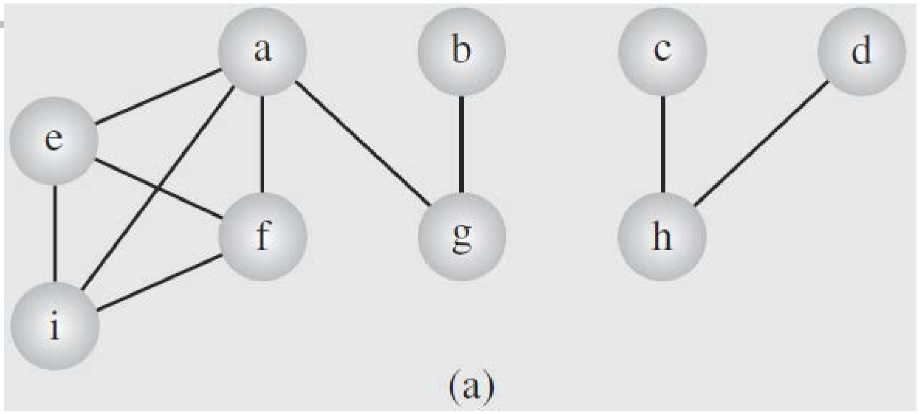




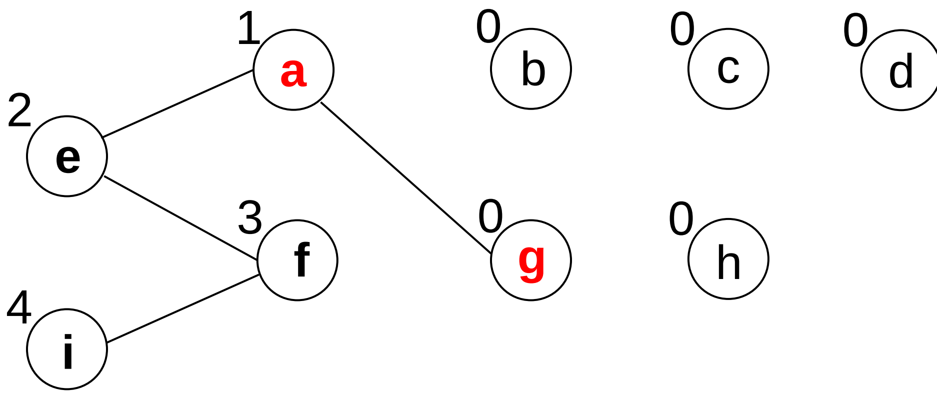
Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```



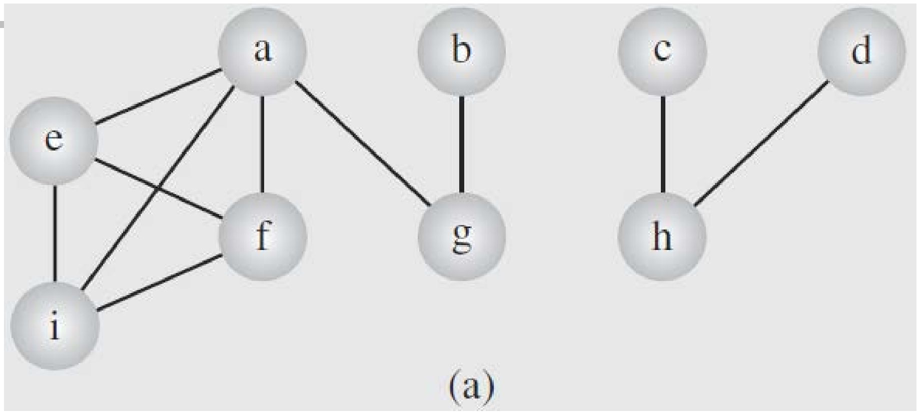
(a)

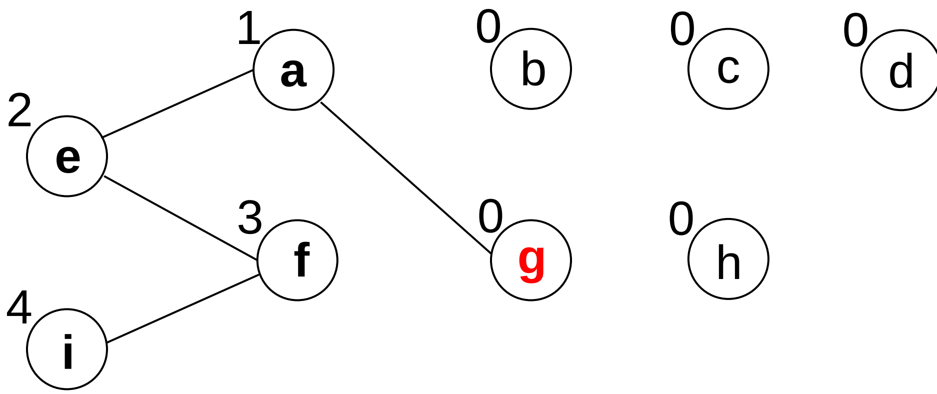


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

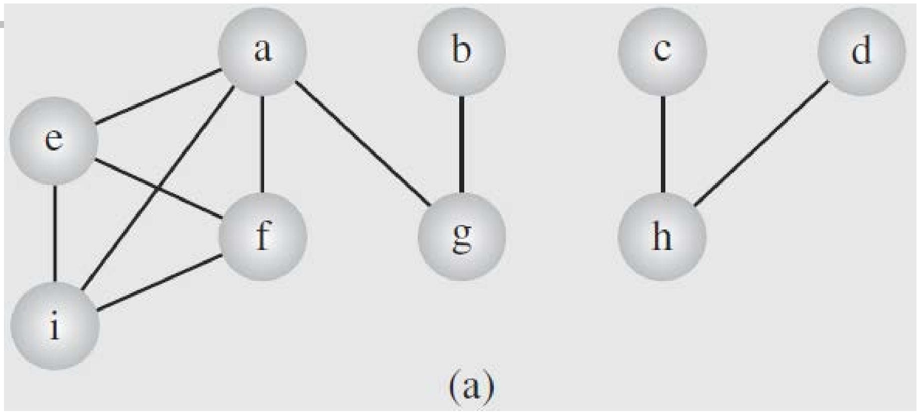


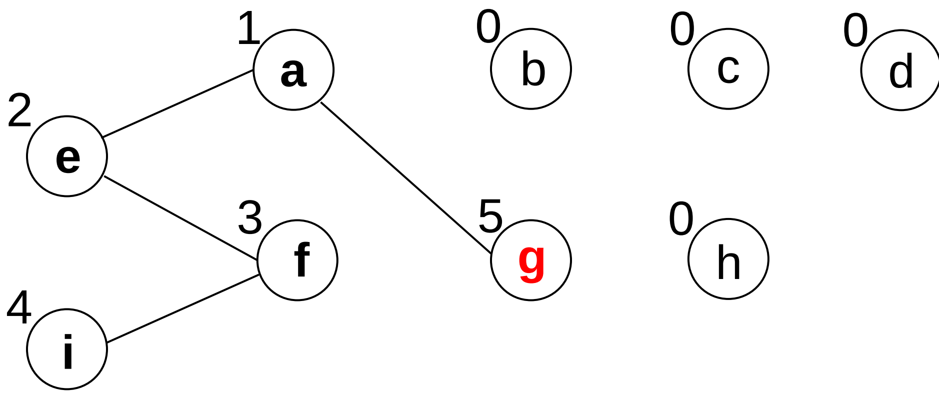


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

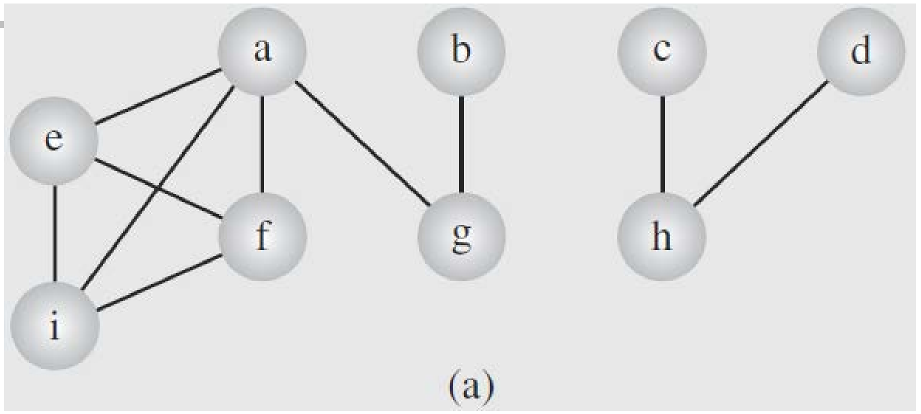




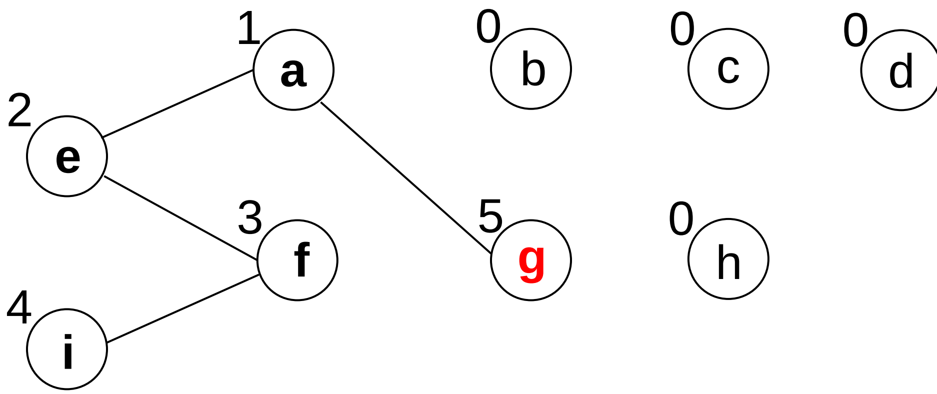
Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```



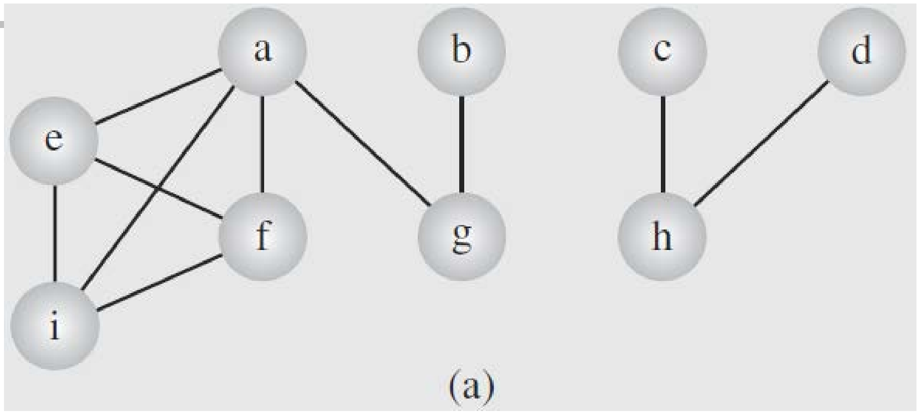
(a)

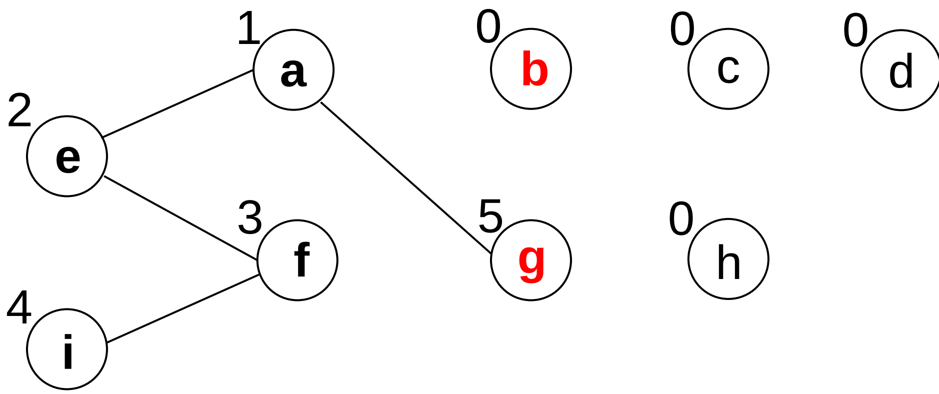


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

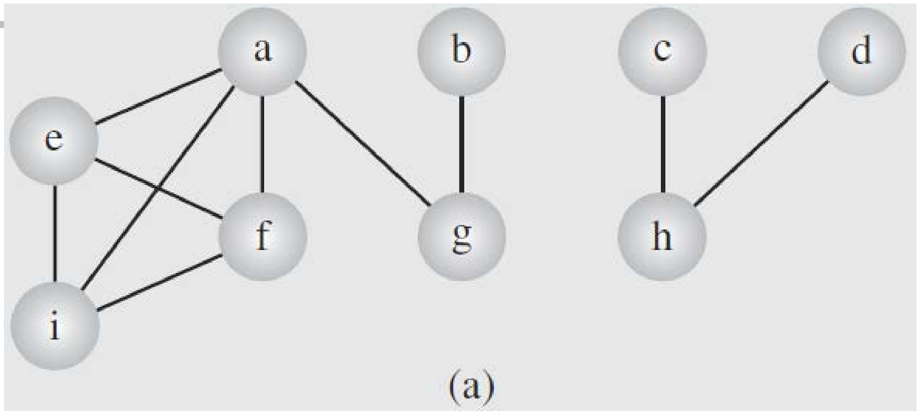


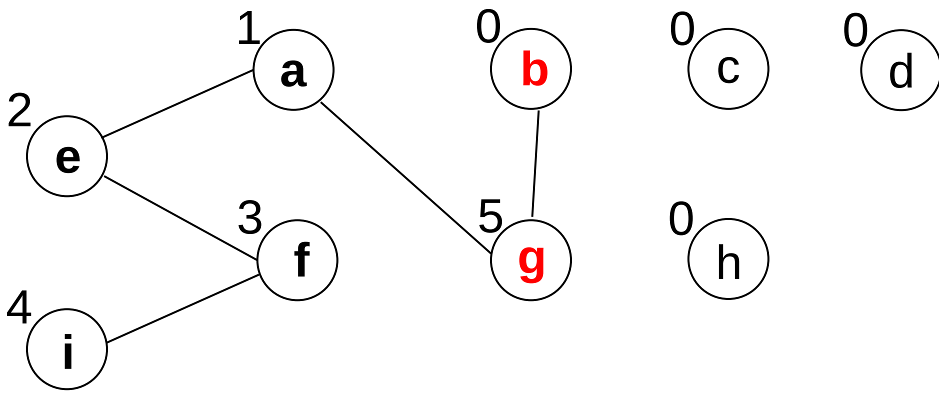


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

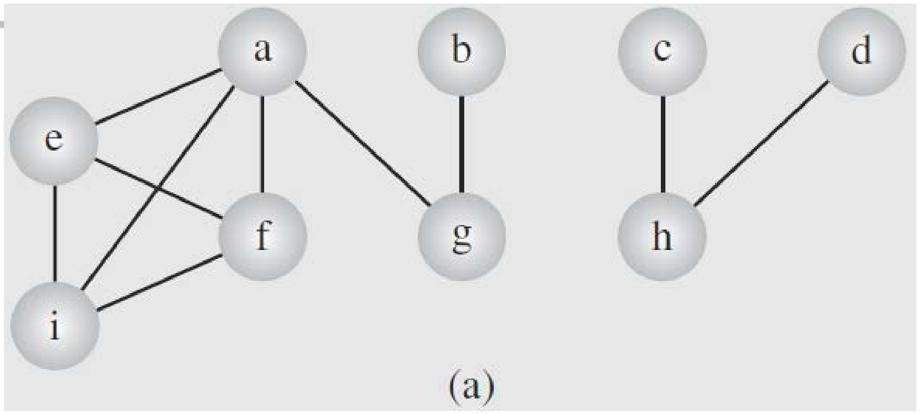




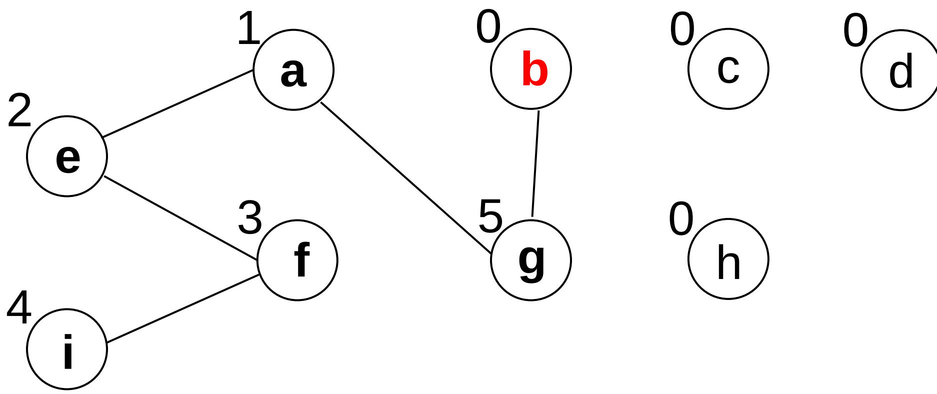
Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```



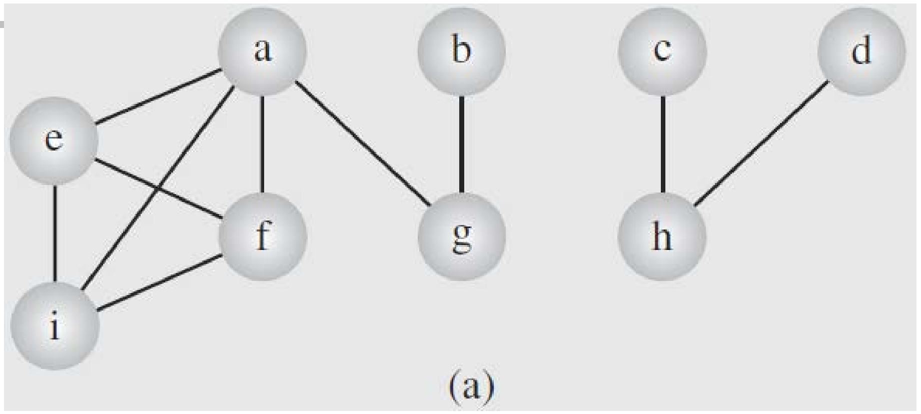
(a)

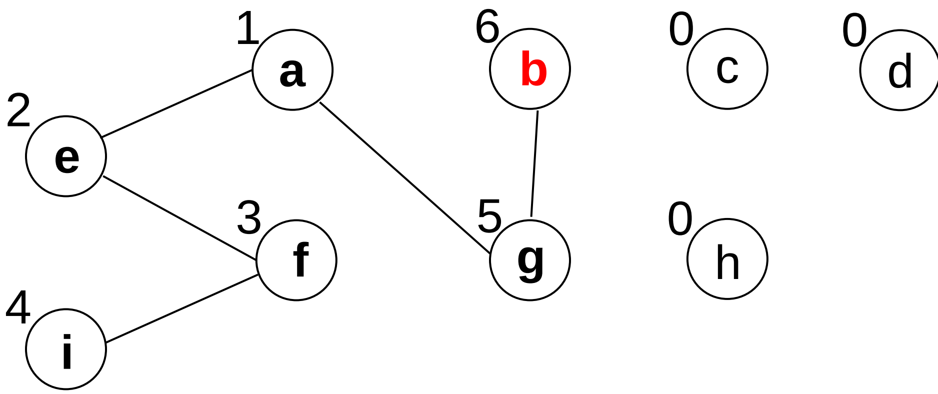


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

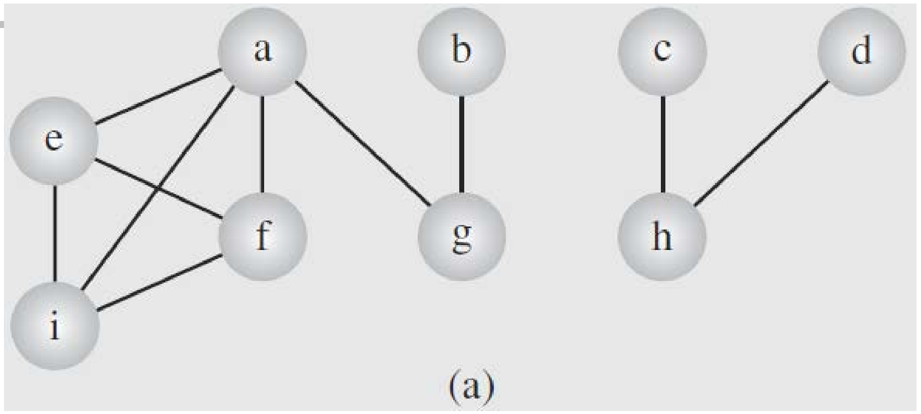


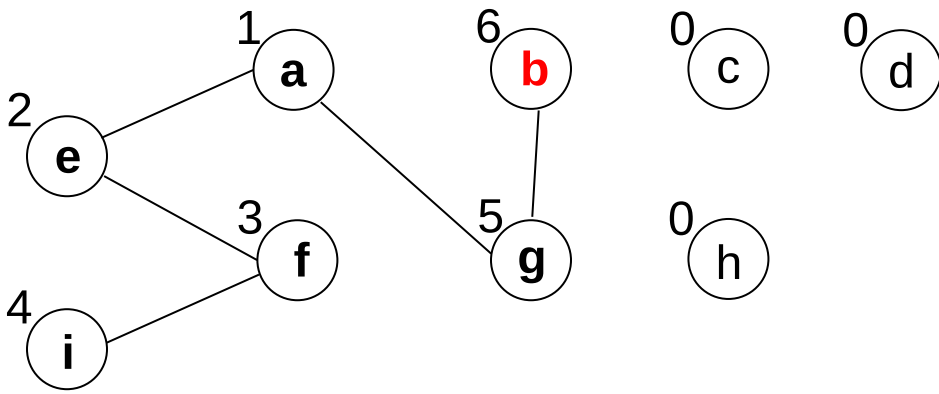


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

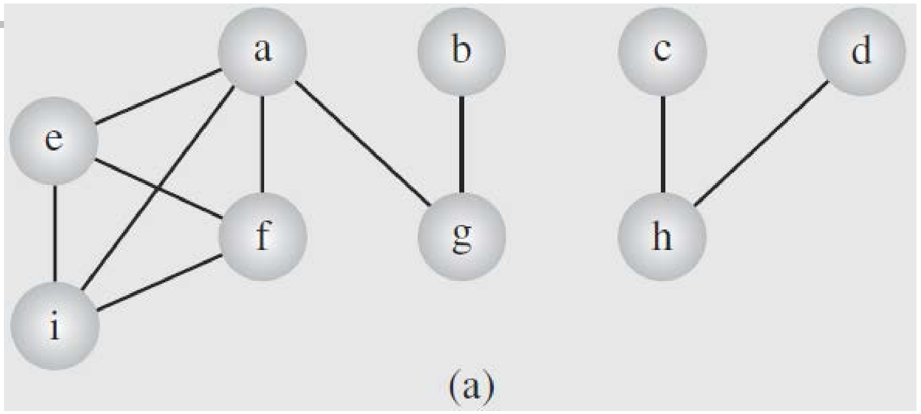


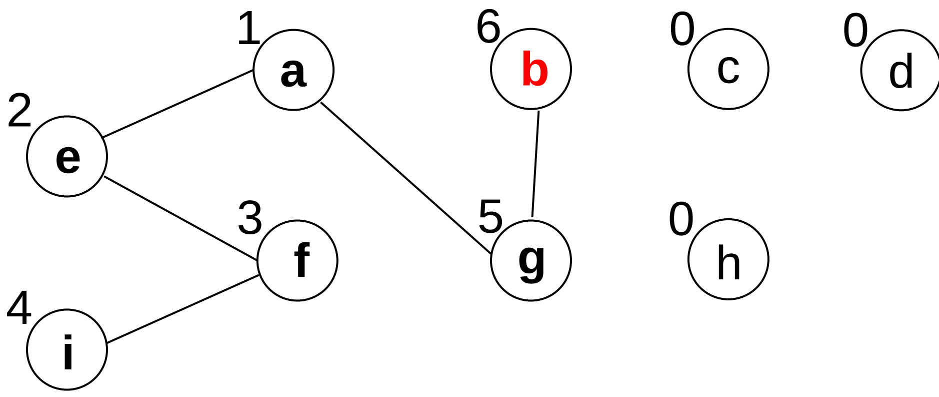


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

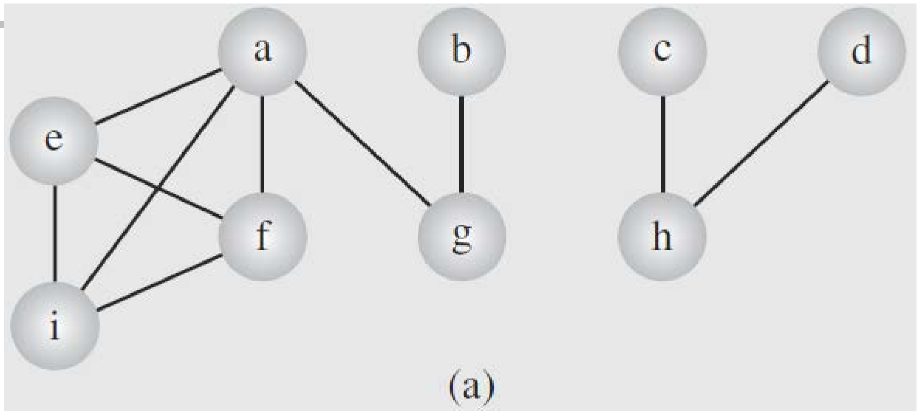


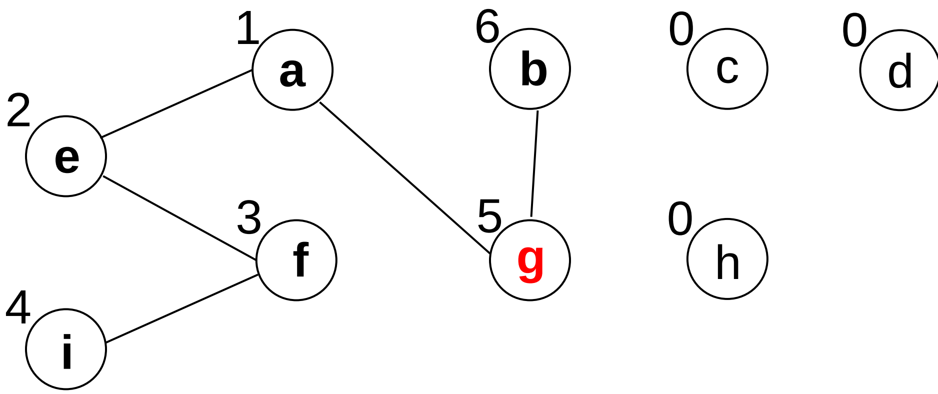


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

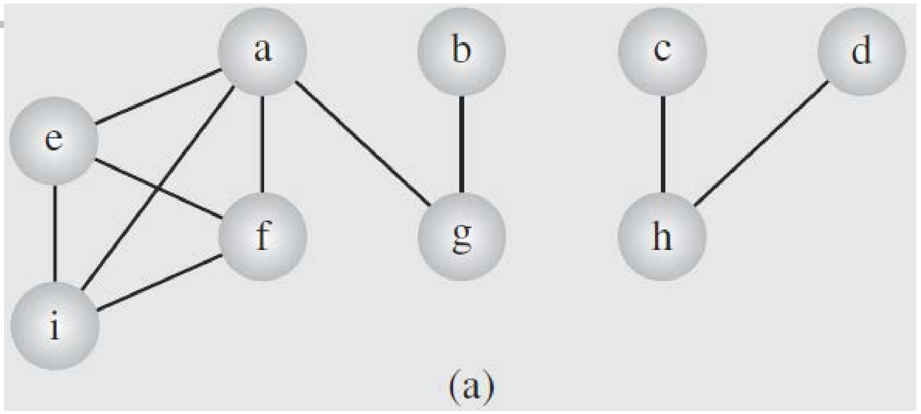


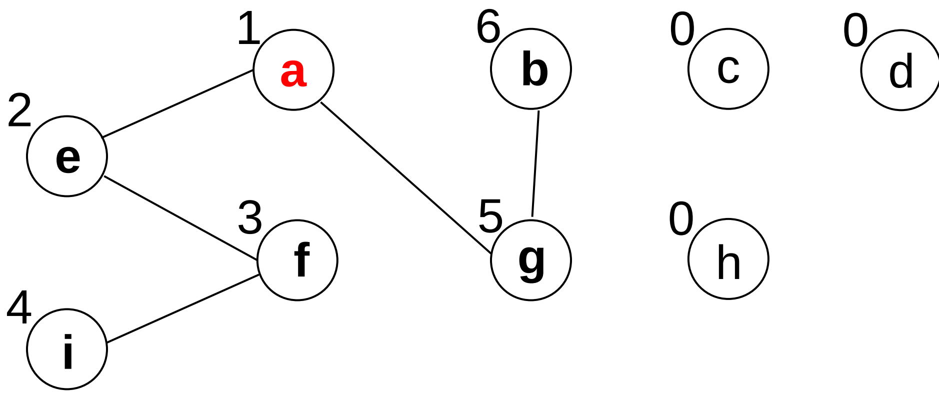


Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```

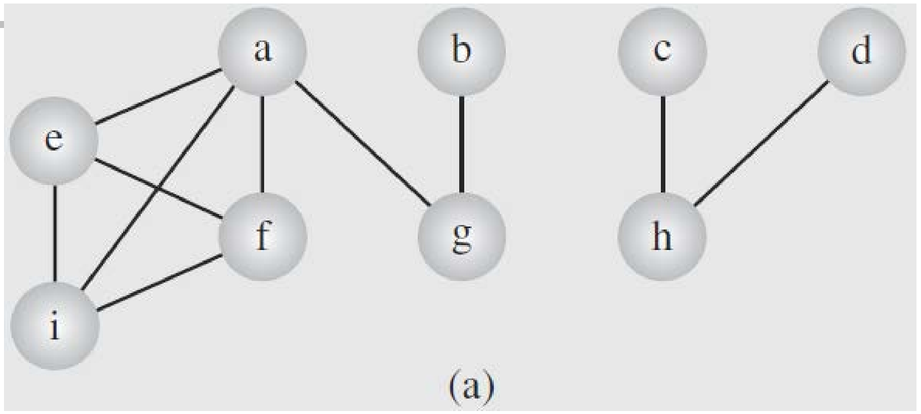




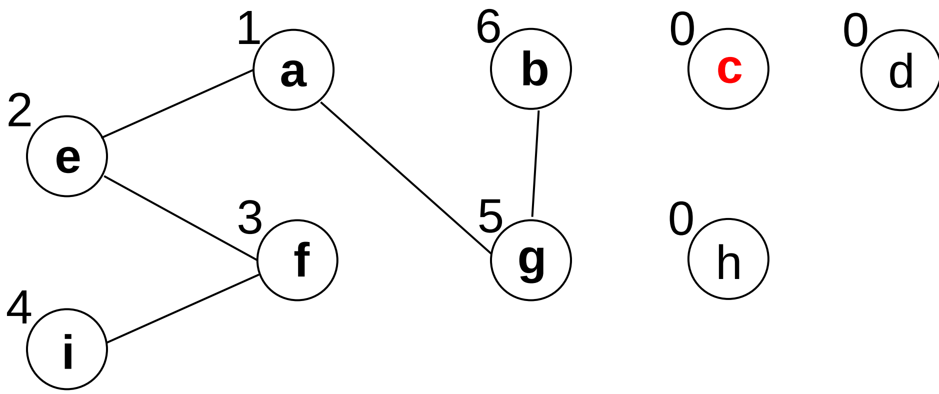
Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```



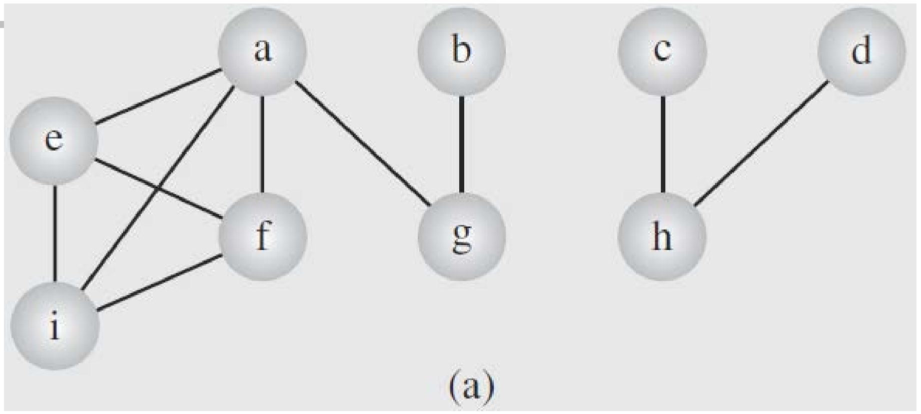
(a)



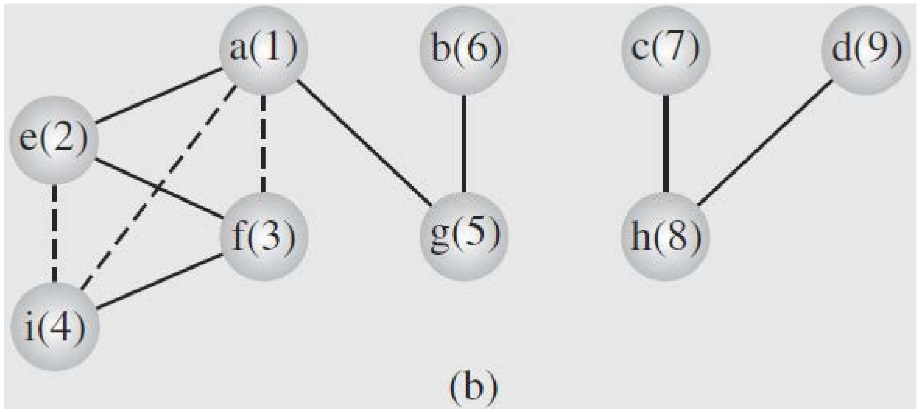
Graph Traversals (cont.)

- **Depth-first search** (cont.),

```
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;
```



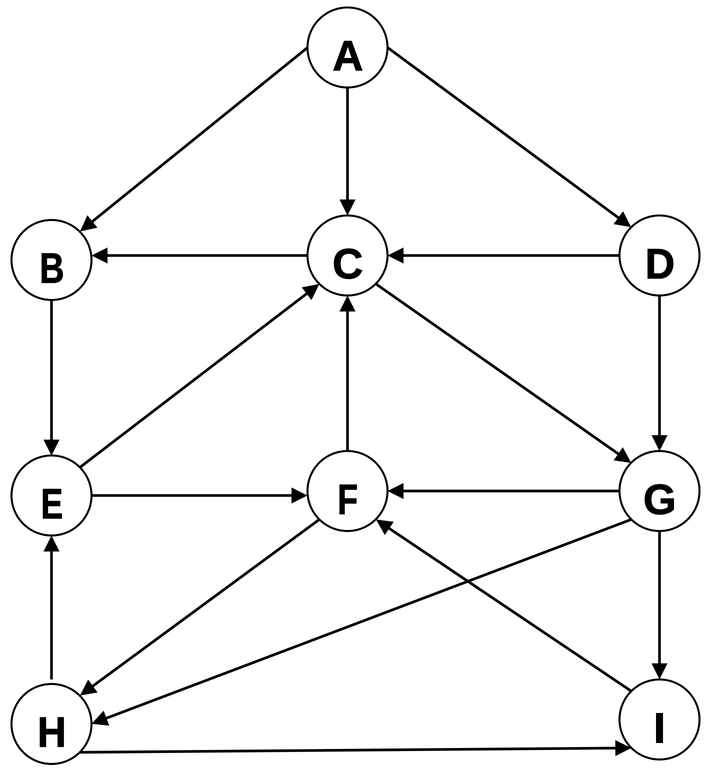
(a)



(b)

Graph Traversals (cont.)

- **Depth-first search** (cont.),
  - example of directed graph – use a **stack**



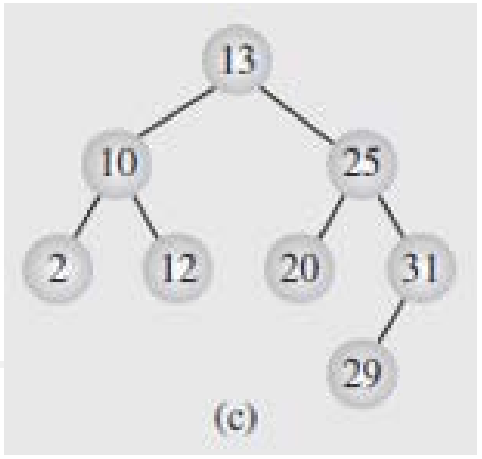
Adjacency Lists			
A:	B	C	D
B:	E		
C:	B	G	
D:	C	G	
E:	C	F	
F:	C	H	
G:	F	H	I
H:	E	I	
I:	F		

Graph Traversals (cont.)

- Recall in tree traversals:
  - depth-first traversal -- use a **stack**
  - breadth-first traversal – use a **queue**

- **Breadth first search**,
  - mark all the vertices accessible from a given vertex, placing them in a **queue** as they are visited
  - the first vertex in the queue is then removed, and the process repeated
  - no visited nodes are revisited
  - if a node has no accessible nodes, the next node in the queue is removed and processed

Tree Traversals: Revisited



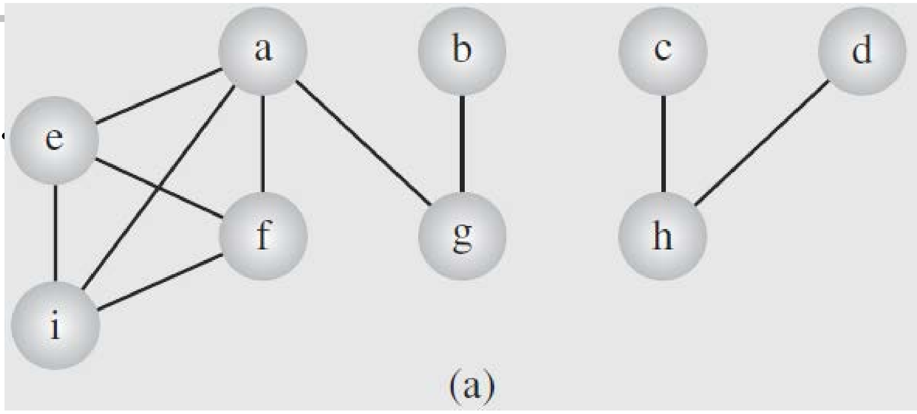
- **Breadth-First Traversal**
  - proceed **level-by-level** from top-down or bottom-up
  - visit each level's nodes left-to-right or right-to-left
  - e.g., 13, 10, 25, 2, 12, 20, 31, 29

- Implement using a **queue**, consider a **top-down**, **left-to-right** breadth-first traversal
  - start by placing the **root node** in the queue
  - then remove the node at the front of the queue
  - **after visiting it**, place its **children** (if any) in the queue
  - repeat until the queue is empty

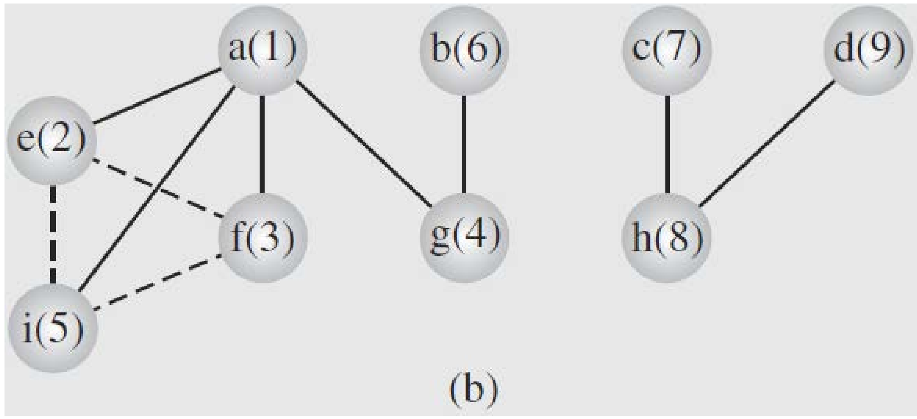
Graph Traversals (cont.)

- **Breadth-First Traversal** (cont.

```
breadthFirstSearch()
    for all vertices u
        num(u) = 0;
    edges = null;
    i = 1;
    while there is a vertex v such that num(v) is 0
        num(v)=i++;
        enqueue(v);
        while queue is not empty
            v = dequeue();
            for all vertices u adjacent to v
                if num(u) is 0
                    num(u) = i++;
                    enqueue(u);
                    attach edge(vu) to edges;
    output edges;
```



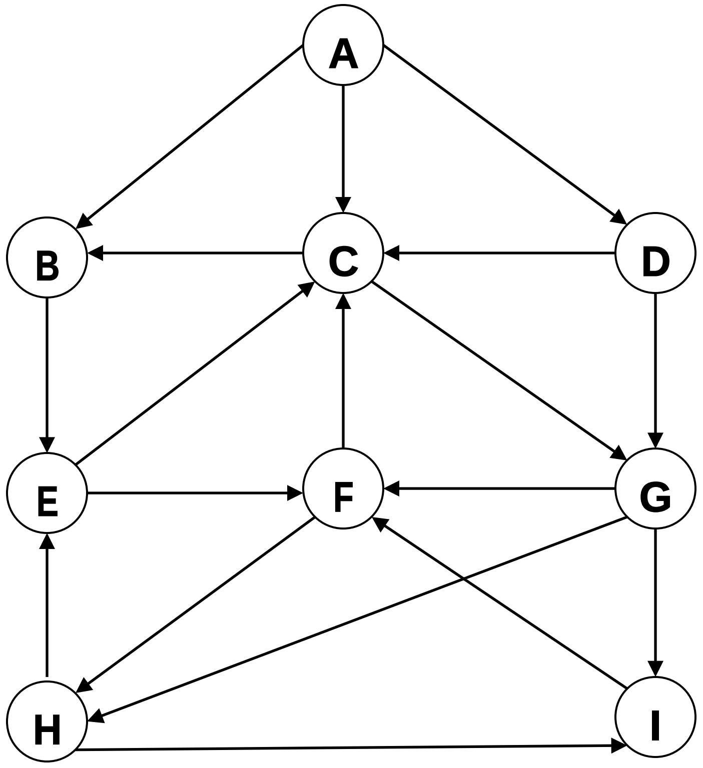
(a)



(b)

Graph Traversals (cont.)

- Breadth first search (cont.),
  - example of directed graph – use a **queue**



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		