Graphs (cont.)

Instructor: Dr. Sunho Lim (Ph.D., Assistant Professor)

Lecture 14

sunho.lim@ttu.edu

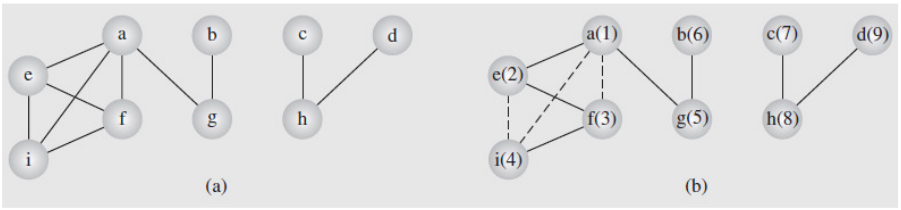
Adapted partially from Data Structures and Algorithms in C++, Adam Drozdek, 4th Edition, Cengage Learning; and Algorithms and Data Structures, Douglas Wilhelm Harder, Mmath

Graph Traversals

- 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 is visited
  - all the unvisited vertices **adjacent** to that vertex are visited
  - If no adjacent vertices, or already visited,
    - **backtrack** to that vertex's predecessor
  - continue until we return to the vertex where the traversal started

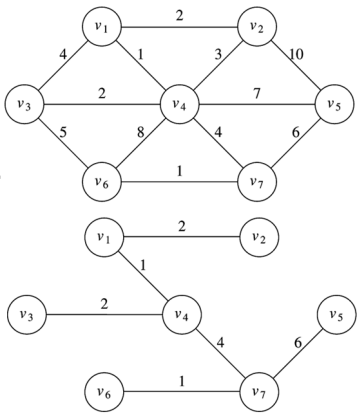
Graph Traversals (cont.)

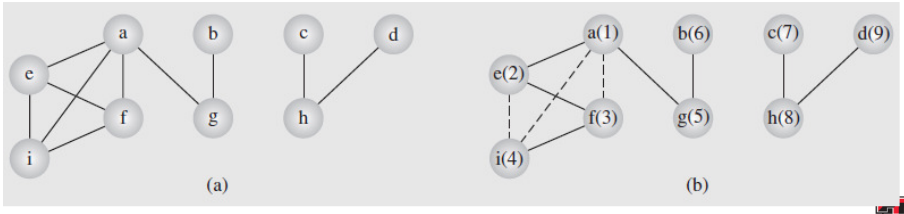
- Depth-first search (cont.),
  - if any vertices remain unvisited at this point,
  - restart the traversal at one of the unvisited vertices
  - e.g., the numbers indicate the order in which the nodes are visited; the solid lines indicate the edges traversed during the search



Graph Traversals (cont.)

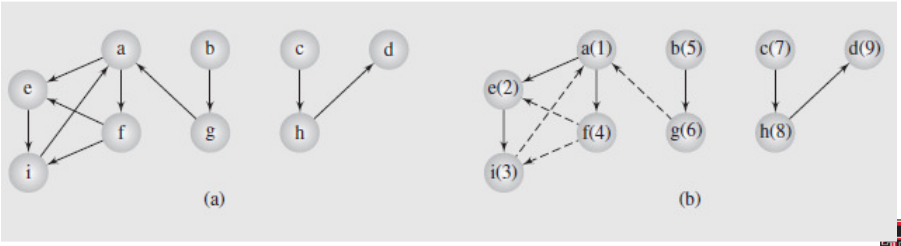
- Depth-first search (cont.),
  - create a tree (or a forest, which is a set of trees) including the graph's vertices, called a **spanning tree**
  - the edges included in the tree are called **forward edges**; those omitted are called **back edges**





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



Tree Traversals: Revisited

- Depth-First Traversal
  - follow the convention of traversing from **left to right**:
    - VLR – known as ***preorder traversal***
    - LVR – known as ***inorder traversal***
    - LRV – known as ***postorder traversal***

```
template<class T>
void BST<T>::preorder(BSTNode<T> *p) {
    if (p != 0) {
        visit(p);
        preorder(p->left);
        preorder(p->right);
    }
}
```

```
template<class T>
void BST<T>::inorder(BSTNode<T> *p) {
    if (p != 0) {
        inorder(p->left);
        visit(p);
        inorder(p->right);
    }
}
```

```
template<class T>
void BST<T>::postorder(BSTNode<T>* p) {
    if (p != 0) {
        postorder(p->left);
        postorder(p->right);
        visit(p);
    }
}
```

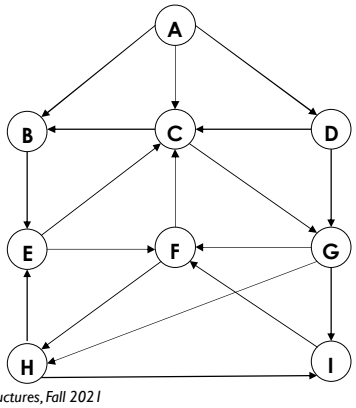
Tree Traversals: Revisited (cont.)

- Depth-First Traversal (continued)
  - the recursion supported by the **run-time stack**
    - a heavy burden on the system
  - e.g., the **inorder** routine
    - traverse the left subtree of the node, then visit the node, then traverse the right subtree

Graph Traversals (cont.)

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

there are nodes reachable from the node H

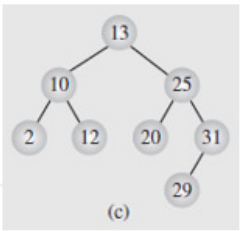


Adjacency Lists			
A:	B	C	D
B:	E		
C:	B	G	
D:	C	G	
E:	C	F	
F:	C	H	
G:	F	H	I
H:	E	I	
I:	F		

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

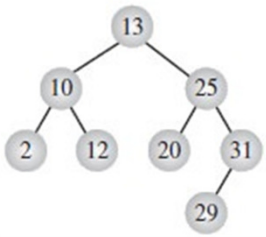
Tree Traversals: Revisited



- **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

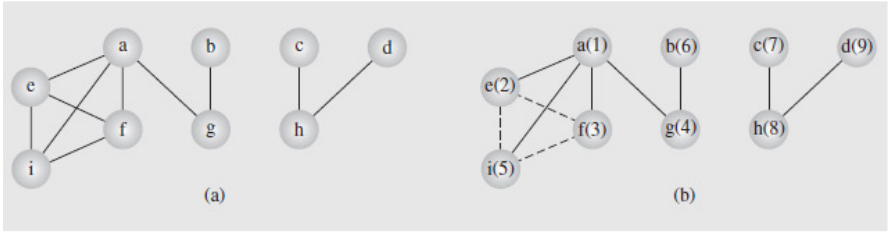
Tree Traversals: Revisited (cont.)

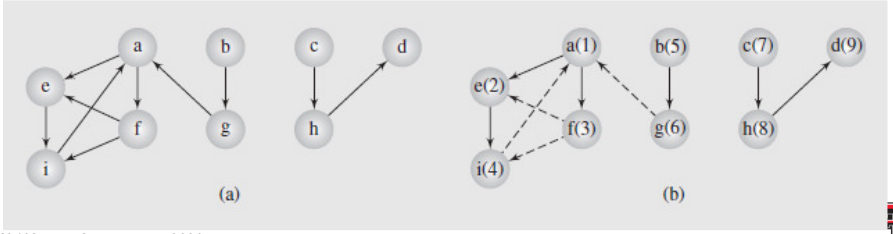
- Breadth-First Traversal (continued)
  - the queue-based breadth-first traversal



Queue				Output
13				
10	25			13
25	2	12		13, 10
2	12	20	31	13, 10, 25
12	20	31		13, 10, 25, 2
20	31			13, 10, 25, 2, 12
31				13, 10, 25, 2, 12, 20
29				13, 10, 25, 2, 12, 20, 31
				13, 10, 25, 2, 12, 20, 31, 29

Graph Traversals (cont.)

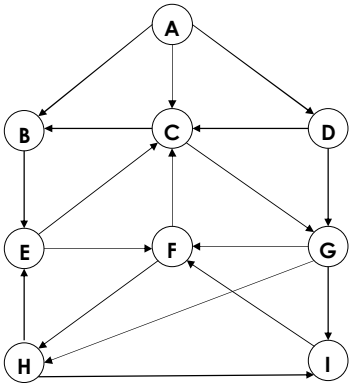




Graph Traversals (cont.)

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

minimum path from A to I, every edge is set to one



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		