CS 395 – Analysis of Algorithms

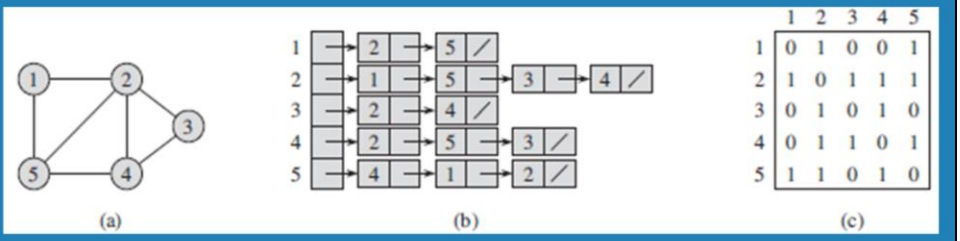
Chapter 3 – Brute Force and Exhaustive Search

Read 3.5

- **Graph Traversal**
  - **Depth-first search (DFS)**
  - **Breadth-first search (BFS)**

Graph Representations

Adjacency Lists and Adjacency Matrices

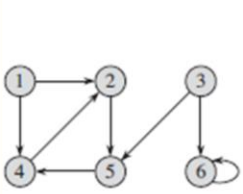


(a) An undirected graph with 5 vertices and 7 (14) edges

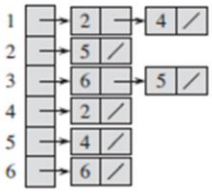
(b) An adjacency-list representation of the graph

(c) The adjacency-matrix representation of the graph

Graph Representations



(a)



(b)

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)

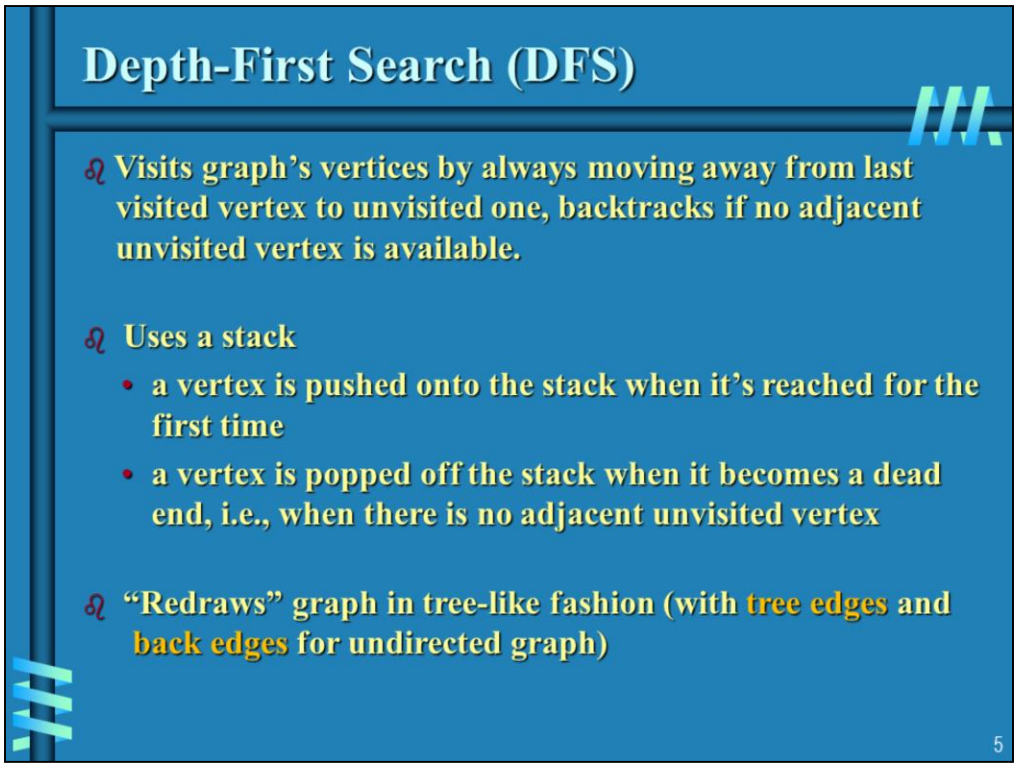
(a) A directed graph with 6 vertices and 8 edges
(b) An adjacency-list representation of the graph
(c) The adjacency-matrix representation of the graph

Graph Traversal Algorithms

Many problems require processing all graph vertices (and edges) in systematic fashion

Graph traversal algorithms:

- Depth-first search (DFS)
- Breadth-first search (BFS)



Depth-first search starts a graph's traversal at an arbitrary vertex by marking it as visited.

On each iteration, the algorithm proceeds to an unvisited vertex that is adjacent to the one it is currentl

- If there are several such vertices, a tie can be resolved arbitrarily.
- As a practical matter, which of the adjacent unvisited candidates is chosen is dictated by the data st
- In our examples, we always break ties by the alphabetical order of the vertices.

This process continues until a dead end—a vertex with no adjacent unvisited vertices—is encountered

At a dead end, the algorithm backs up one edge to the vertex it came from and tries to continue visitin

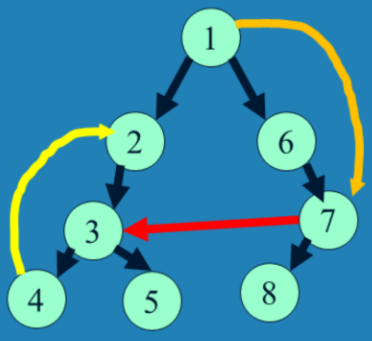
The algorithm eventually halts after backing up to the starting vertex, with the latter being a dead end.

By then, all the vertices in the same connected component as the starting vertex have been visited.

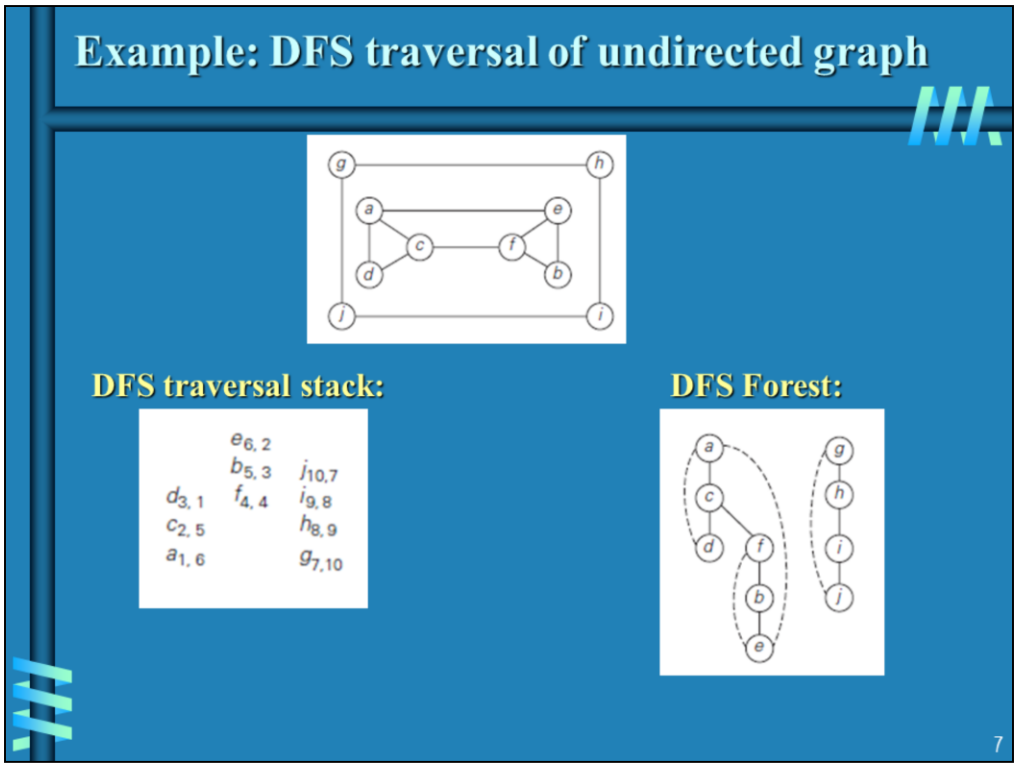
If unvisited vertices still remain, the depth-first search must be restarted at any one of them.

Preview of some terms

Tree Edge
Cross Edge
Back Edge
Forward Edge



6



Example of a DFS traversal.

(a) Graph.

(b) Traversal's stack

- the first subscript number indicates the order in which a vertex is visited, i.e., pushed onto the stack
- the second one indicates the order in which it becomes a dead-end, i.e., popped off the stack).

(c) DFS forest with the tree and back edges shown with solid and dashed lines, respectively.

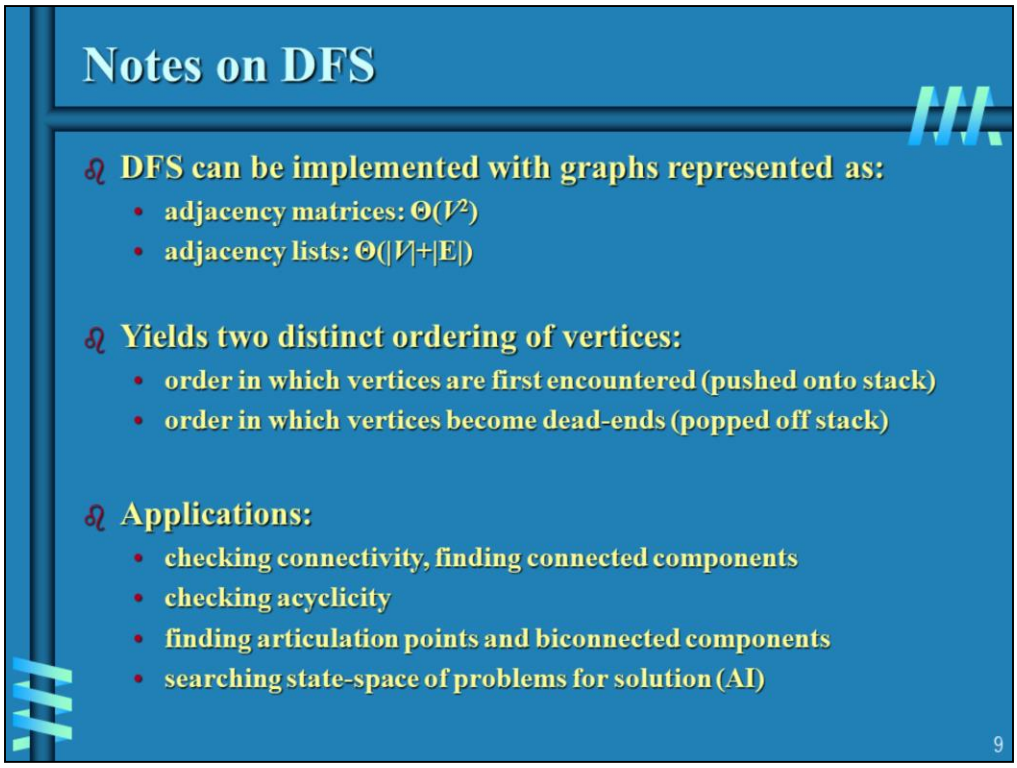
- A back edge is an edge leading to a previously visited vertex other than its immediate predecessor

Pseudocode of DFS

ALGORITHM *DFS*(G)

```
//Implements a depth-first search traversal of a given graph
//Input: Graph G = ⟨V, E⟩
//Output: Graph G with its vertices marked with consecutive integers
//in the order they've been first encountered by the DFS traversal
mark each vertex in V with 0 as a mark of being "unvisited"
count ← 0
for each vertex v in V do
    if v is marked with 0
        dfs(v)

dfs(v)
//visits recursively all the unvisited vertices connected to vertex v by a path
//and numbers them in the order they are encountered
//via global variable count
count ← count + 1; mark v with count
for each vertex w in V adjacent to v do
    if w is marked with 0
        dfs(w)
```

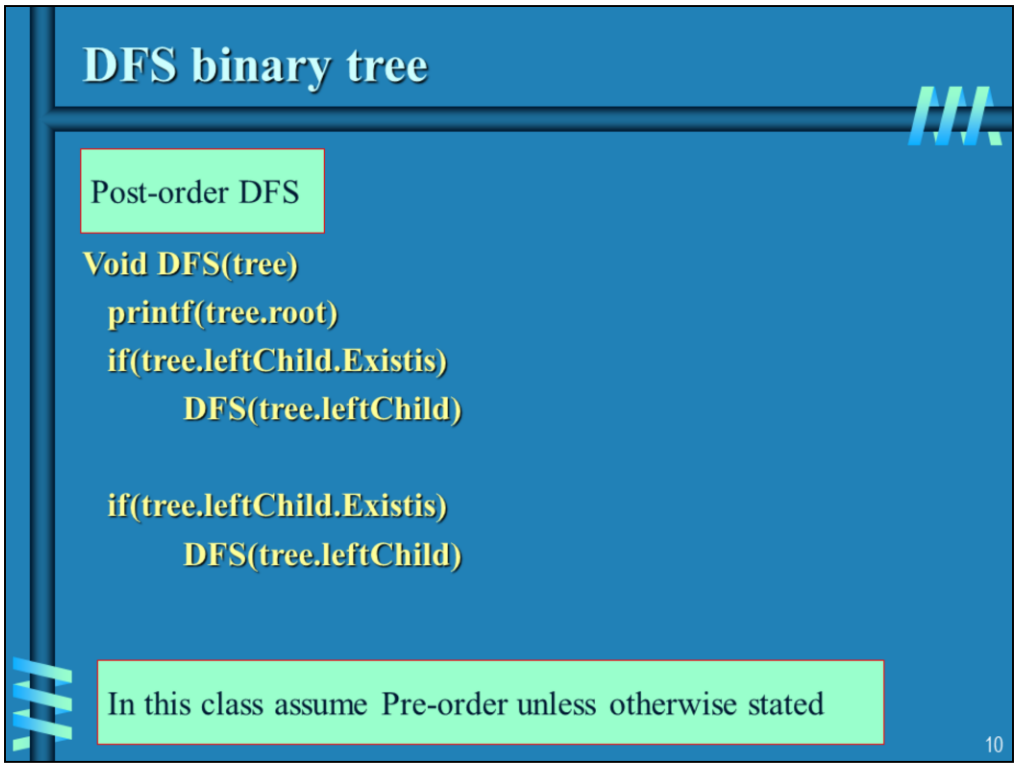


adjacency lists: $\Theta(|V|+|E|)$ is more efficient since you only need to check nodes that are actually conne
In a fully connected graph $|E| = V(V-1)/2 = \Theta(V^2)$

A vertex of a connected graph is said to be its ***articulation point*** if its removal with all edges incident

- If any child of a node does not have a path to any of the ancestors of its parent, it is an ***articulation***

A biconnected graph has no articulation points



adjacency lists: $\Theta(|V|+|E|)$ is more efficient since you only need to check nodes that are actually conne
In a fully connected graph $|E| = V(V-1)/2 = \Theta(V^2)$

A vertex of a connected graph is said to be its ***articulation point*** if its removal with all edges incident

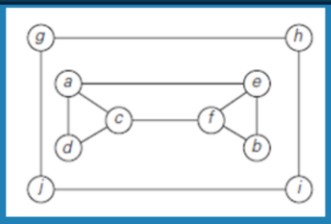
- If any child of a node does not have a path to any of the ancestors of its parent, it is an ***articulation***

A biconnected graph has no articulation points

Breadth-first search (BFS)

- Visits graph vertices by moving across to all the neighbors of last visited vertex
- Instead of a stack, BFS uses a queue
- Similar to level-by-level tree traversal
- "Redraws" graph in tree-like fashion (with tree edges and cross edges for undirected graph)

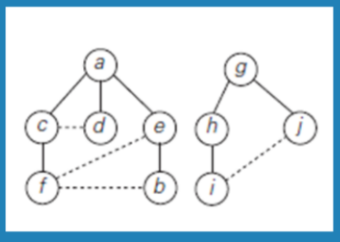
Example: BFS traversal of undirected graph



BFS traversal queue:

$a_1\, c_2\, d_3\, e_4\, f_5\, b_6$
$g_7\, h_8\, j_9\, i_{10}$

BFS Forest:



12

Pseudocode of BFS

```
ALGORITHM BFS(G)
    //Implements a breadth-first search traversal of a given graph
    //Input: Graph G = (V, E)
    //Output: Graph G with its vertices marked with consecutive integers
    //in the order they have been visited by the BFS traversal
    mark each vertex in V with 0 as a mark of being "unvisited"
    count ← 0
    for each vertex v in V do
        if v is marked with 0
            bfs(v)

    bfs(v)
    //visits all the unvisited vertices connected to vertex v by a path
    //and assigns them the numbers in the order they are visited
    //via global variable count
    count ← count + 1;   mark v with count and initialize a queue with v
    while the queue is not empty do
        for each vertex w in V adjacent to the front vertex do
            if w is marked with 0
                count ← count + 1;   mark w with count
                add w to the queue
        remove the front vertex from the queue
```

Notes on BFS

- BFS has same efficiency as DFS and can be implemented with graphs represented as:
  - adjacency matrices: $\Theta(|V|^2)$
  - adjacency lists: $\Theta(|V|+|E|)$
- Yields single ordering of vertices (order added/deleted from queue is the same)
- Applications: same as DFS, but can also find paths from a vertex to all other vertices with the smallest number of edges

DFS and BFS Summary

	DFS	BFS
Data structure	a stack	a queue
Number of vertex orderings	two orderings	one ordering
Edge types (undirected graphs)	tree and back edges	tree and cross edges
Applications	connectivity, acyclicity, articulation points	connectivity, acyclicity, minimum-edge paths
Efficiency for adjacency matrix	$\Theta(\|V^2\|)$	$\Theta(\|V^2\|)$
Efficiency for adjacency lists	$\Theta(\|V\| + \|E\|)$	$\Theta(\|V\| + \|E\|)$