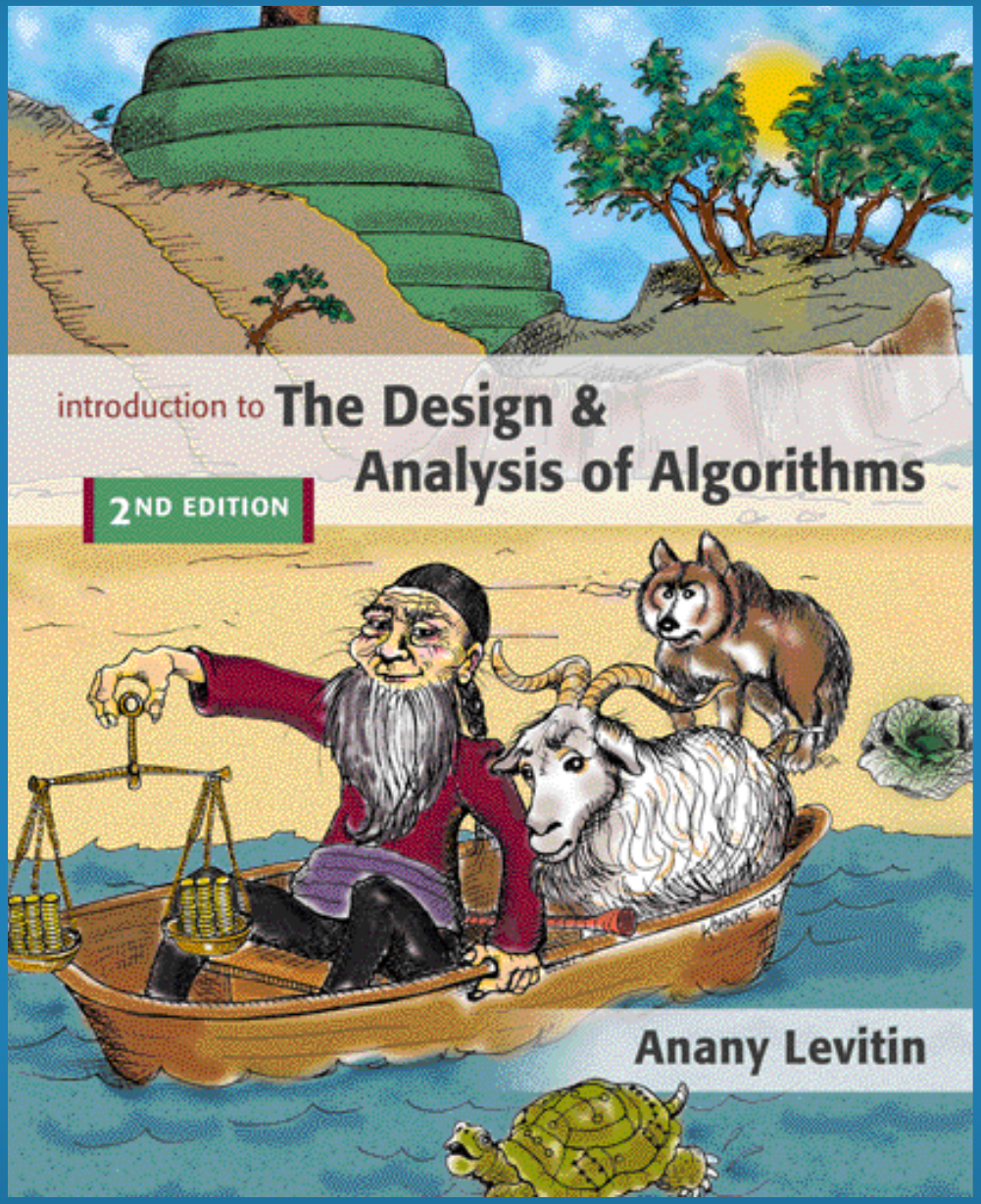
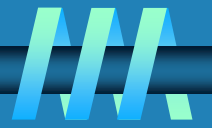
Chapter 5

Decrease-and-Conquer

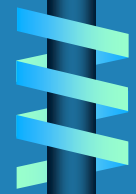


Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

Decrease-and-Conquer

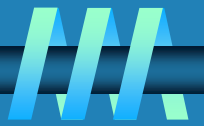


1. Reduce problem instance to smaller instance of the same problem
2. Solve smaller instance
3. Extend solution of smaller instance to obtain solution to original instance



Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

3 Types of Decrease and Conquer



- ***Decrease by a constant*** **(usually by 1):**
  - **insertion sort**
  - **graph traversal algorithms (DFS and BFS)**
  - **topological sorting**
  - **algorithms for generating permutations, subsets**

- ***Decrease by a constant factor*** **(usually by half)**
  - **binary search**

- ***Variable-size decrease***
  - **Euclid's algorithm**

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

Insertion Sort

To sort array A[0..*n*-1], sort A[0..*n*-2] recursively and then insert A[*n*-1] in its proper place among the sorted A[0..*n*-2]

- **Usually implemented bottom up (nonrecursively)**

Example: Sort 6, 4, 1, 8, 5

6 | 4 1 8 5
4 6 | 1 8 5
1 4 6 | 8 5
1 4 6 8 | 5
1 4 5 6 8

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

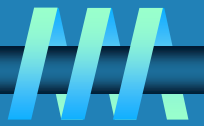
Pseudocode of Insertion Sort

ALGORITHM *InsertionSort*$(A[0..n-1])$

```
//Sorts a given array by insertion sort
//Input: An array A[0..n − 1] of n orderable elements
//Output: Array A[0..n − 1] sorted in nondecreasing order
for i ← 1 to n − 1 do
    v ← A[i]
    j ← i − 1
    while j ≥ 0 and A[j] > v do
        A[j + 1] ← A[j]
        j ← j − 1
    A[j + 1] ← v
```

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

Analysis of Insertion Sort



- **Time efficiency**

  $C_{worst}(n) = n(n\text{-}1)/2 \in \Theta(n^2)$

  $C_{avg}(n) \approx n^2/4 \in \Theta(n^2)$

  $C_{best}(n) = n - 1 \in \Theta(n)$ (also fast on almost sorted arrays)

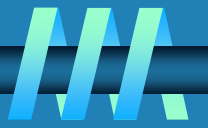
- **Space efficiency: in-place**

- **Stability: yes**

- **Best elementary sorting algorithm overall**

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

Graph Traversal



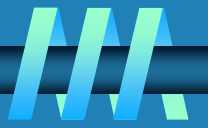
Many problems require processing all graph vertices (and edges) in systematic fashion

<u>Graph traversal algorithms</u>:

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

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

Depth-First Search (DFS)



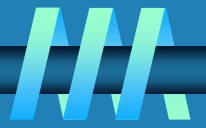
- Visits graph's vertices by always moving away from last visited vertex to an unvisited one, backtracks if no adjacent unvisited vertex is available.

- Recurisve or it uses a stack
  - a vertex is pushed onto the stack when it's reached for the first time
  - a vertex is popped off the stack when it becomes a dead end, i.e., when there is no adjacent unvisited vertex

- "Redraws" graph in tree-like fashion (with tree edges and back edges for undirected graph)

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

Pseudocode of DFS



ALGORITHM *DFS*(G)

//Implements a depth-first search traversal of a given graph
//Input: Graph $G = \langle V, E \rangle$
//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"
$count \leftarrow 0$
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*
$count \leftarrow count + 1$; mark v with *count*
for each vertex w in V adjacent to v **do**
    if w is marked with 0
        dfs(w)

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

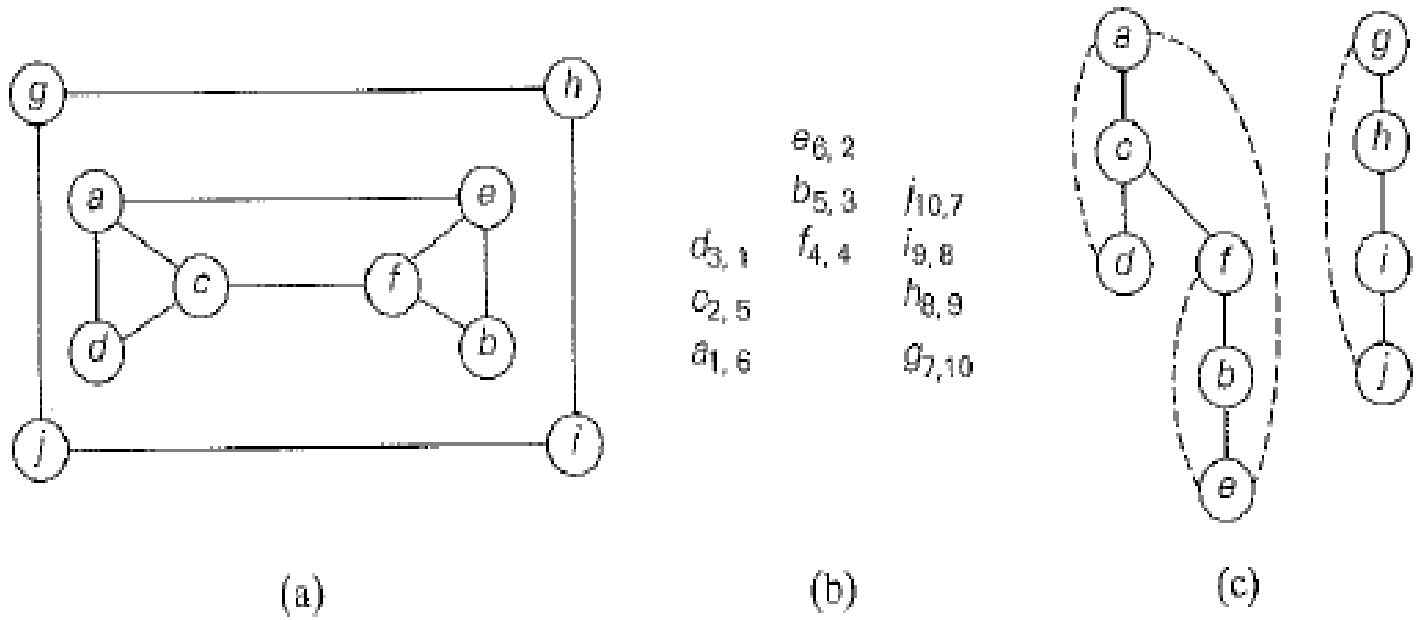
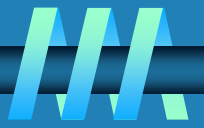


FIGURE 5.5 Example of a DFS traversal. (a) Graph. (b) Traversal's stack (the first subscript number indicates the order in which a vertex was visited, i.e., pushed onto the stack; the second one indicates the order in which it became a dead-end, i.e., popped off the stack). (c) DFS forest (with the tree edges shown with solid lines and the back edges shown with dashed lines).

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

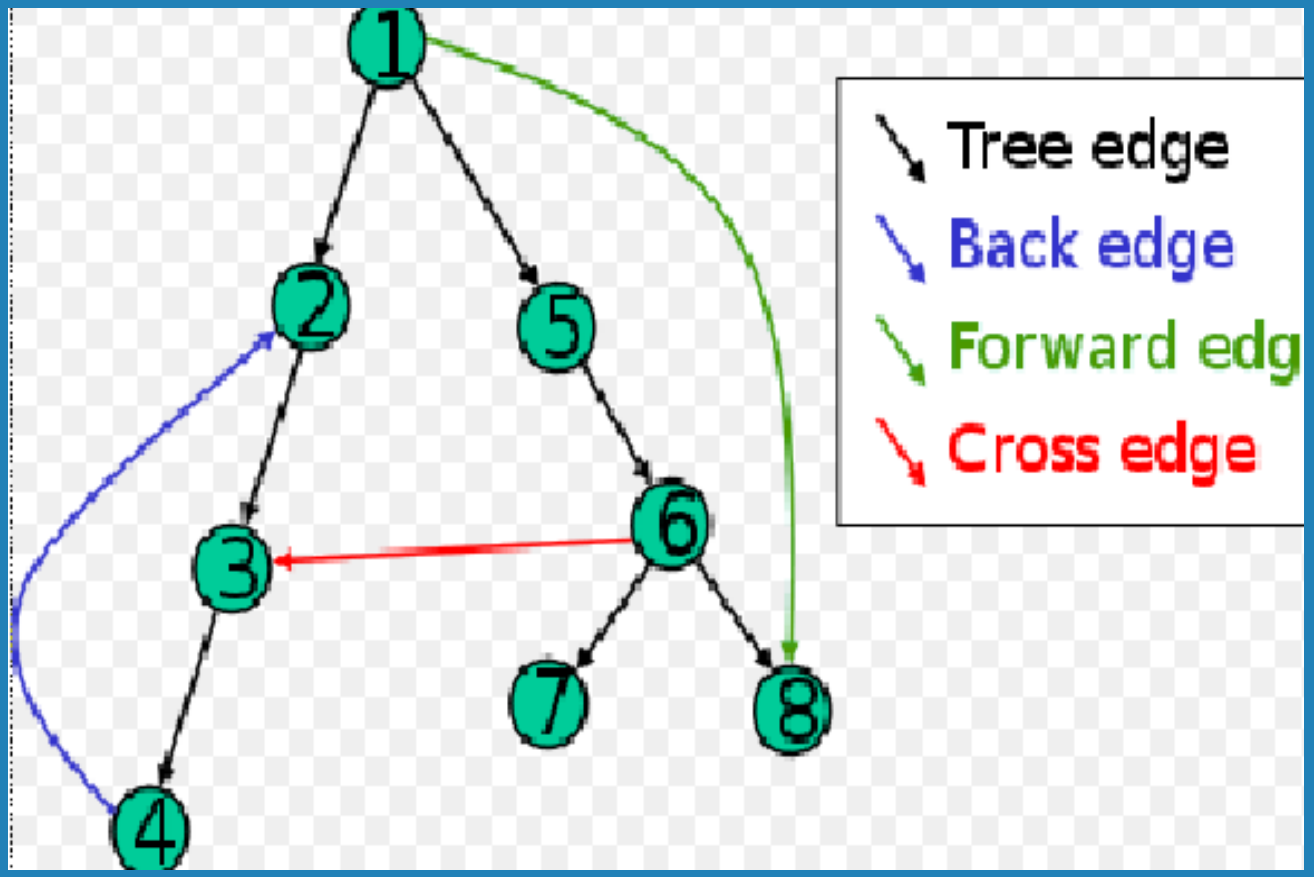
Notes on DFS



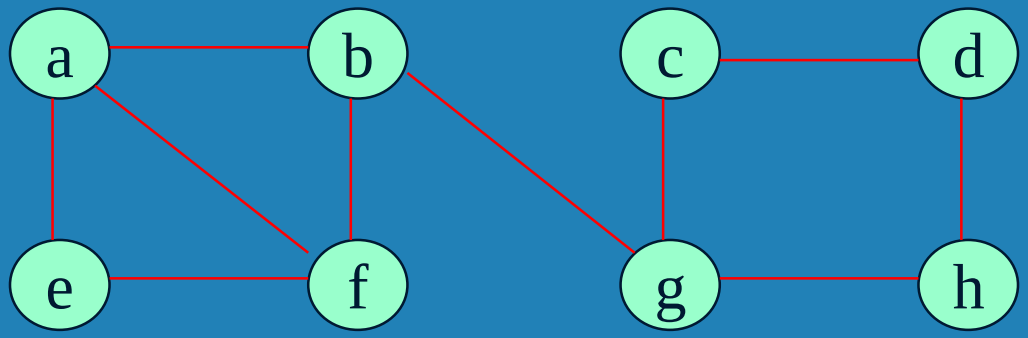
- DFS can be implemented with graphs represented as:
  - adjacency matrices: $\Theta(|V|^2)$.
  - adjacency lists: $\Theta(|V|+|E|)$.
- Yields two distinct ordering of vertices:
  - order in which vertices are first encountered (pushed onto stack)
  - order in which vertices become dead-ends (popped off stack)
- Applications:
  - checking connectivity, finding connected components
  - checking acyclicity (if no back edges)
  - finding articulation points

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

Contd…

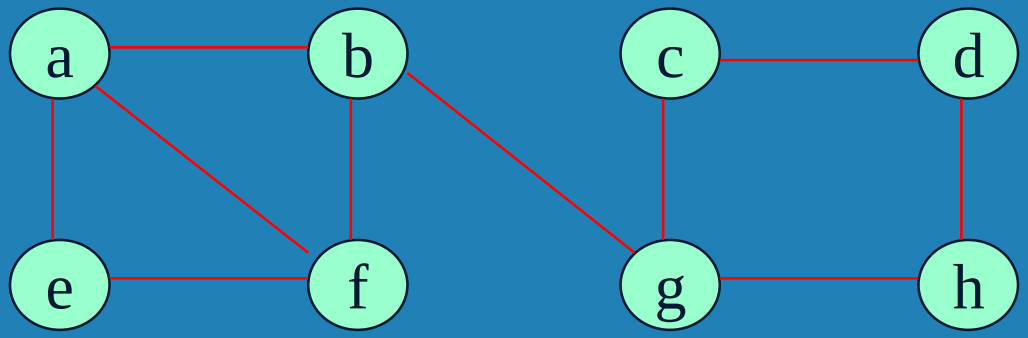


Copyright © 2007 Pearson Addison-Wesley. All rights reserved.



Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

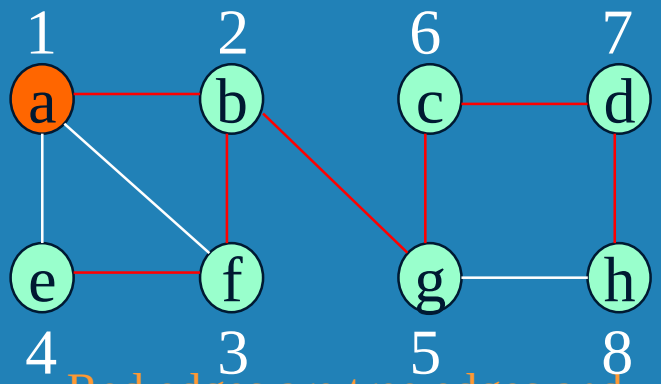
Example: DFS traversal of undirected graph



DFS traversal stack:

a
ab
abf
abfe
abf
ab
abg
abgc
abgcd
abgcdh
abgcd
:

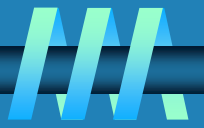
DFS tree:



Red edges are tree edges and white edges are back edges.

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

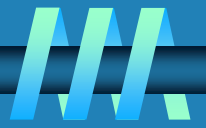
Breadth-first search (BFS)



- Visits graph vertices by moving across to all the neighbors of the 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)

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

Pseudocode of BFS

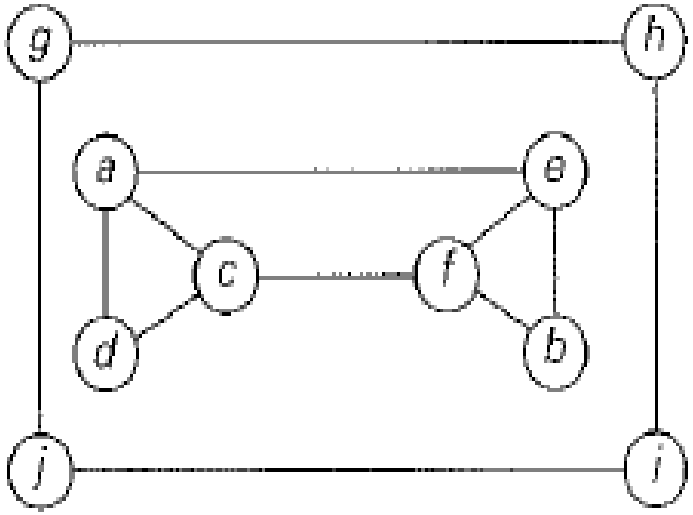


ALGORITHM *BFS*(G)

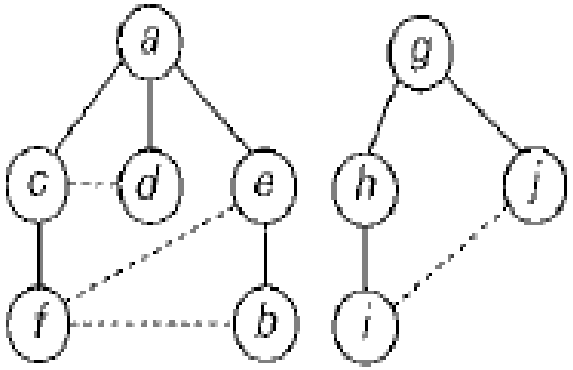
```
//Implements a breadth-first search traversal of a given graph
//Input: Graph G = ⟨V, E⟩
//Output: Graph G with its vertices marked with consecutive integers
//in the order they have been visited by the BFS traversal
mark each vertex in V with 0 as a mark of being "unvisited"
count ← 0
for each vertex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

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.



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

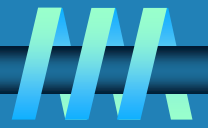


(a) (b) (c)

FIGURE 5.6 Example of a BFS traversal. (a) Graph. (b) Traversal's queue, with the numbers indicating the order in which the vertices were visited, i.e., added to (or removed from) the queue. (c) BFS forest (with the tree edges shown with solid lines and the cross edges shown with dotted lines).

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

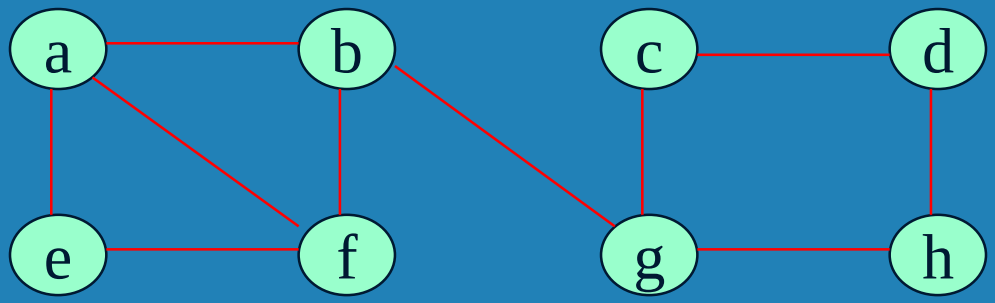
Notes on BFS



- 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

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

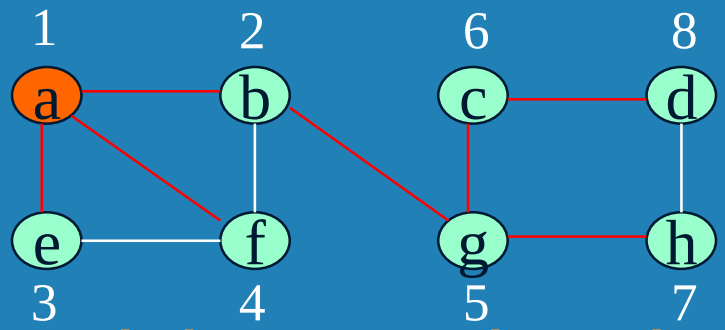
Example of BFS traversal of undirected graph



BFS traversal queue:

a
bef
efg
fg
g
ch
hd
d

BFS tree:



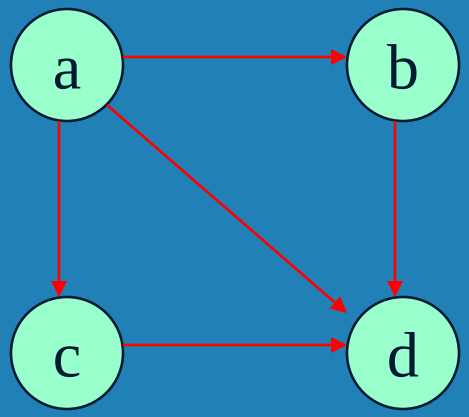
Red edges are tree edges and white edges are cross edges.

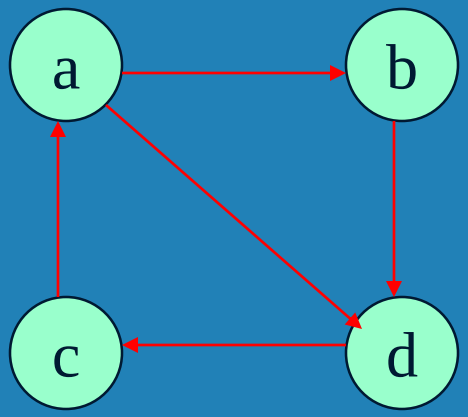
Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

DAGs and Topological Sorting

A *dag*: a directed acyclic graph, i.e. a directed graph with no (directed) cycles

a dag





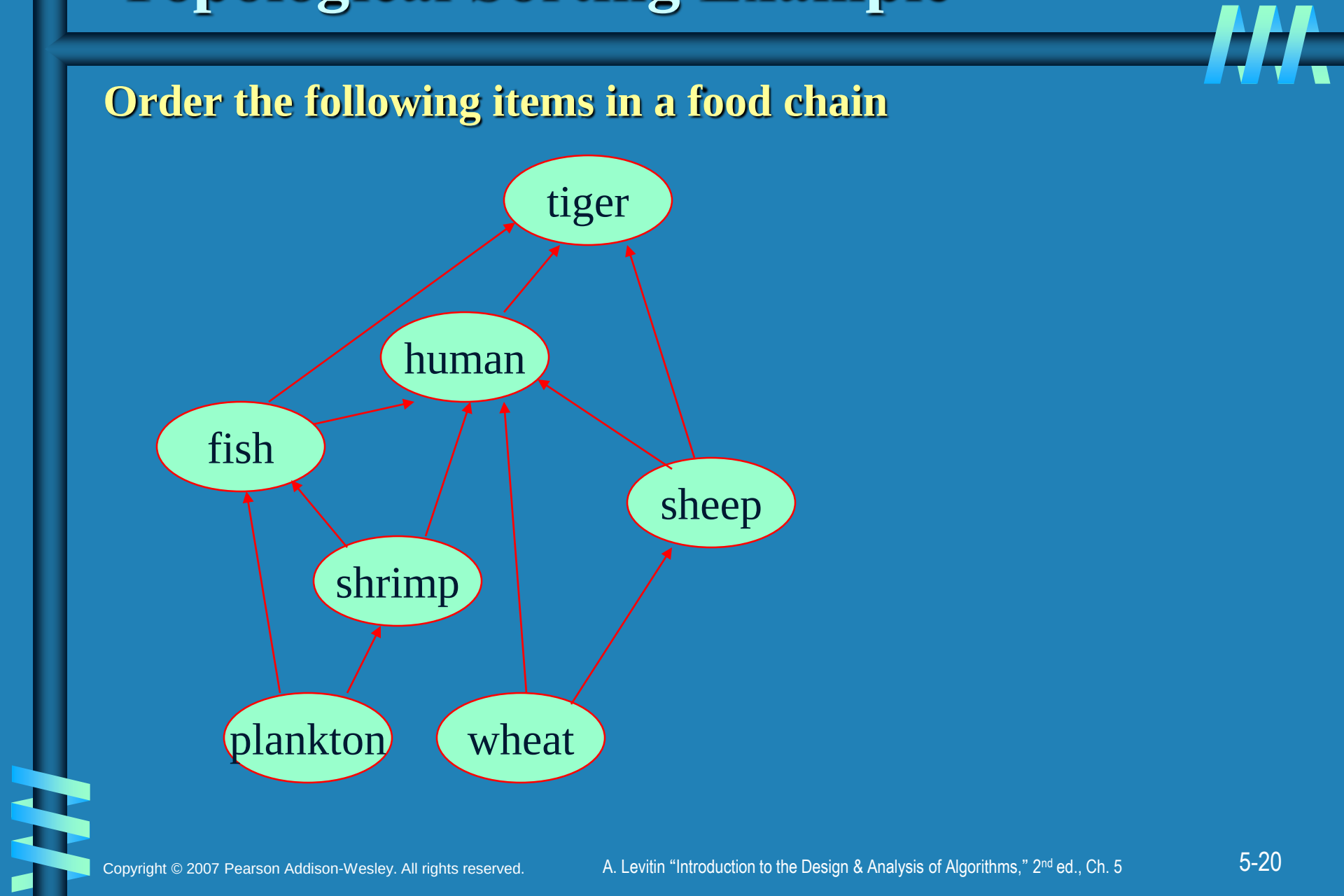
not a dag

Arise in modeling many problems that involve prerequisite constraints (construction projects, document version control)

Vertices of a dag can be linearly ordered so that for every edge its starting vertex is listed before its ending vertex (*topological sorting*). Being a dag is also a necessary condition for topological sorting to be possible.

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

Topological Sorting Example

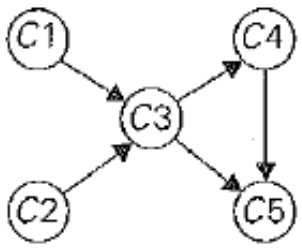


Order the following items in a food chain

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

DFS-based algorithm for topological sorting

- Perform DFS traversal, noting the order vertices are popped off the traversal stack
- Reverse order solves topological sorting problem
- Back edges encountered?→ NOT a dag!



(a)

$C5_1$
$C4_2$
$C3_3$
$C1_4$ $C2_5$

(b)

The popping-off order:
C5, C4, C3, C1, C2
The topologically sorted list:
C2 C1→C3→C4→C5

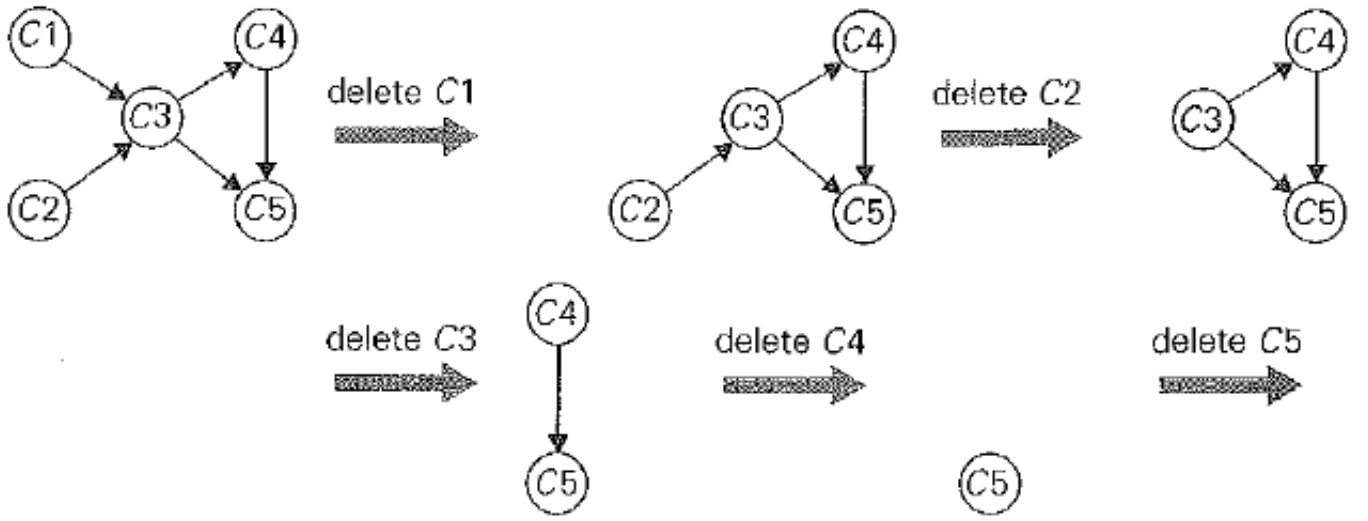
(c)

FIGURE 5.10 (a) Digraph for which the topological sorting problem needs to be solved. (b) DFS traversal stack with the subscript numbers indicating the popping-off order. (c) Solution to the problem.

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

Source removal algorithm

Repeatedly identify and remove a *source* (a vertex with no incoming edges) and all the edges incident to it until either no vertex is left or there is no source among the remaining vertices (not a dag)

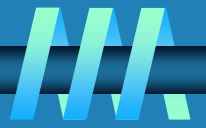


The solution obtained is C1, C2, C3, C4, C5

FIGURE 5.11 Illustration of the source-removal algorithm for the topological sorting problem. On each iteration, a vertex with no incoming edges is deleted from the digraph.

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

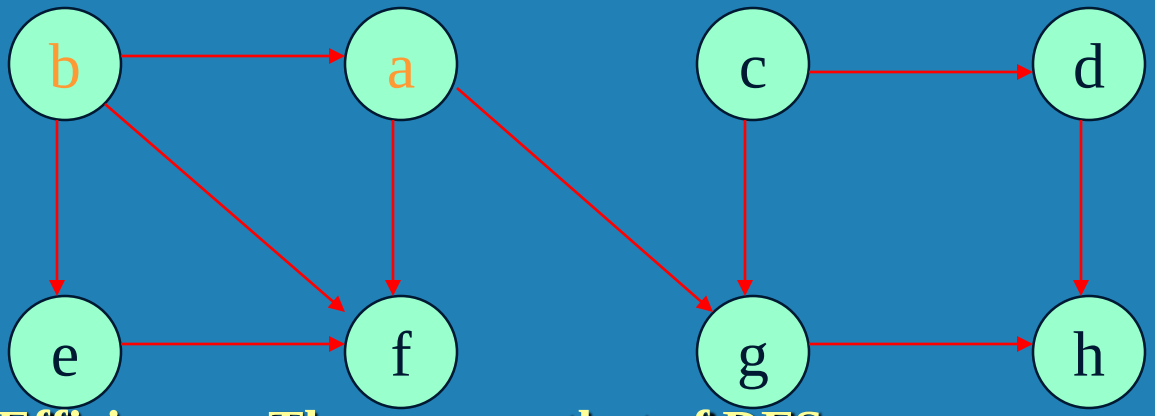
DFS-based Algorithm



DFS-based algorithm for topological sorting

- **Perform DFS traversal, noting the order vertices are popped off the traversal stack**
- **Reverse order solves topological sorting problem**
- **Back edges encountered?→ NOT a dag!**

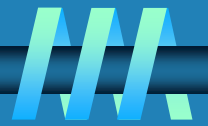
Example:



Efficiency: The same as that of DFS.

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.

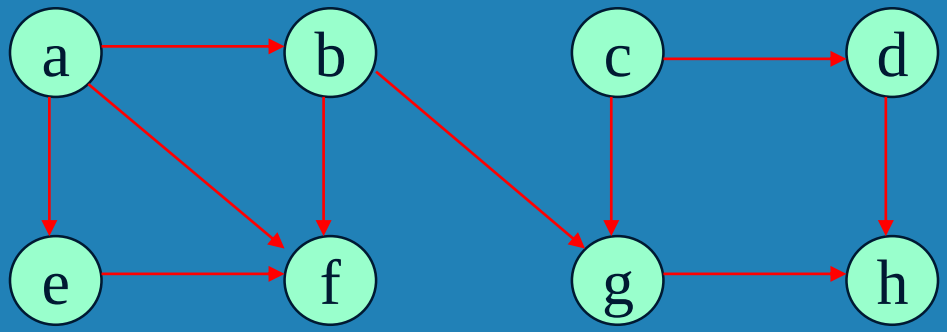
Source Removal Algorithm



Source removal algorithm

Repeatedly identify and remove a *source* (a vertex with no incoming edges) and all the edges incident to it until either no vertex is left or there is no source among the remaining vertices (not a dag)

Example:



Efficiency: same as efficiency of the DFS-based algorithm.

Copyright © 2007 Pearson Addison-Wesley. All rights reserved.