Class 07

GRAPH SEARCH

DEPTH-FIRST SEARCH

BREADTH-FIRST SEARCH

TOPOLOGICAL SORT

Student Learning Objectives

Students should be able to…

- Traverse a graph using breadth-first search and depth-first search
- Conduct DFS graph traversal with labeling
- Sort nodes of a directed acyclic graph (dag) topologically using DFS and source-removal

Graph Traversal

Exhaustive search of a graph: visit every vertex/edge

Two key approaches:
- Depth-first search
- Breadth-first search

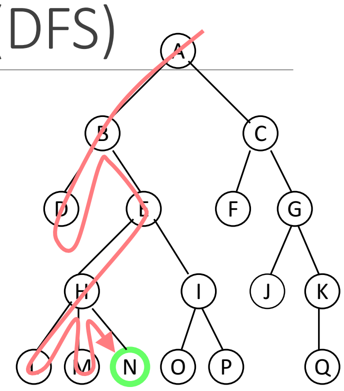
Searching a graph will be represented in form of a tree

Let *n* be the number of vertices,

Let *m* be the number of edges

Depth-First Search (DFS)

- Search trees form a forest.
- $T(n) \in \Theta(m+n)$
- Applications: checking connectivity, acyclicity; spanning tree



Depth-First Search (DFS)

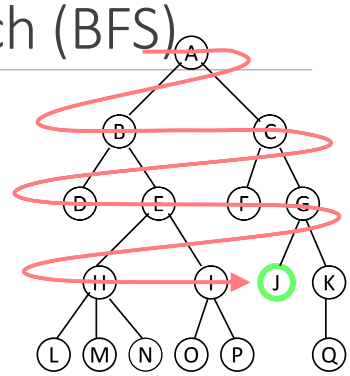
ALGORITHM *DFS(G)*

//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)

Breadth-First Search (BFS)

- Search trees form a forest.
- $T(n) \in \Theta(m+n)$
- Applications: shortest paths (unweighted), DFS applications, etc.



Breadth-First Search (BFS)

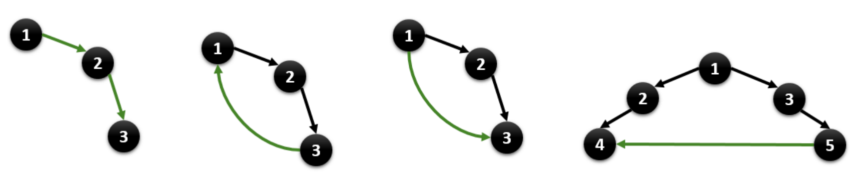
```
ALGORITHM  BFS(G)
    //Implements a breadth-first search traversal of a given graph
    //Input: Graph G = ⟨V, E⟩
    //Output: Graph G with its vertices marked with consecutive integers
    //in the order they have been visited by the BFS traversal
    mark each vertex in V with 0 as a mark of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

Edge Types

For a **directed** graph (**digraph**), only traverse forward along edges.

Edges are categorized by the traversal into four types:



Tree edge Back edge Forward edge Cross edge

Images from http://alexvolov.com/2015/02/depth-first-search-dfs/

Topological Sort

Problem: Given a directed acyclic graph (dag), order the vertices so that for all edges (i,j), i is before j.

Consider the following example.

One solution is: C1, C2, C3, C4, C5

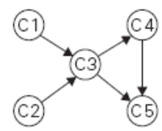


FIGURE 4.6 Digraph representing the prerequisite structure of five courses.

Topological Sorting Algorithms

Algorithm 1. DFS-based

- Run the DFS on the dag and output the vertices in reverse order of finishing time.

Algorithm 2. Source-removal

- Iteratively remove a "source" (in-degree = 0) from the graph. Removal order → topological sort.

Topological Sort
DFS-based

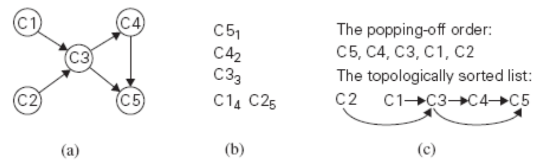


FIGURE 4.7 (a) Digraph for which the topological sorting problem needs to be solved. (b) DFS traversal stack with the subscript numbers indicating the popping-off order. (c) Solution to the problem.

Topological Sort
Source Removal

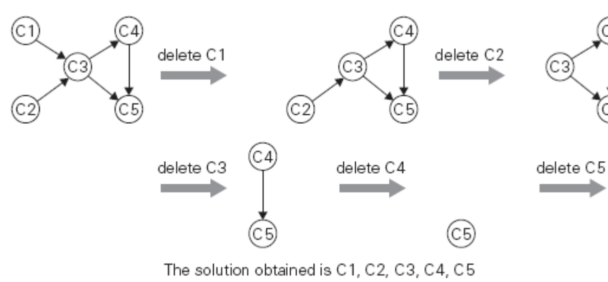


FIGURE 4.8 Illustration of the source-removal algorithm for the topological sorting problem. On each iteration, a vertex with no incoming edges is deleted from the digraph.