Graph Traversal: Breadth First Search

Strong Connectivity

- Strongly Connected: A graph for which there is a directed path from any node to any other node

- Is this graph strongly connected?

- Strongly connected component: A strongly connected sub-graph

- Can you find the strongly connected components of this graph?
The Web is a Graph

- Directed Graph of Nodes and Arcs
  - Nodes = web pages
  - Arcs = hyperlinks from a page to another

- A graph can be explored
- A graph can be indexed

Traversing the Web

- The web can be considered a graph “the web graph”
- Web pages are the graph nodes
- Hyperlinks on pages are graph arcs
- The web graph is huge (way over one million billions nodes) - maybe infinite (pages are created on the fly)
- For traversing the web graph, DFS is not a good strategy. (Why?)
Breadth First Example: BFS(9)

BFS(9) pseudocode

// BFS traversal starting at v
// Initialization:
Mark all vertices as unvisited
enQueue v onto a new queue Q
Mark v as visited
// Procedure:
While (Q is not empty)
    deQueue a vertex w from Q
    For each unvisited vertex u adjacent to w:
        enQueue u onto Q
        Mark u as visited

BFS from S to G:
The BFS tree shows the visits
How do you remember the path?
Initialization: enqueue path [S] in Q
While you have not reached G
dequeue a path from BFS queue and
check the last node x in the path
extend the path to unvisited neighbors of x
and enqueue extended paths to back of Q

How do you remember the path?
Dependency Graph on a DAG

- Defined on a Directed Acyclic Graph (a “DAG”)
- Usually reflect dependencies or requirements
  - I.e., Assembly lines, Supply lines, Organizational charts, ...
  - BTW: You cannot take 231 after 230 unless...
- Understanding dependencies requires “topological sorting”

Resolving DAG Dependencies

- Topological order
  - A list of vertices in a DAG such that vertex $x$ precedes vertex $y$ iff there is a directed edge from $x$ to $y$ in the graph
  - There may be several topological orders in a given graph
- Topological sorting
  - Arranging the vertices into a topological order
Topological Sorting Algorithm

- Select a vertex $v$ that has **no predecessor**
- Remove $v$ from the graph (along with all associated arcs),
- Add $v$ to the end of a list of vertices $L$
- Repeat previous steps
- When the graph is empty, $L$’s vertices will be in topological order

A(another) Topological Sorting Algorithm

- Select a vertex $v$ that has **no successor**
- Remove $v$ from the graph (along with all associated arcs),
- Add $v$ to the beginning of a list of vertices $L$
- Repeat previous steps
- When the graph is empty, $L$’s vertices will be in topological order
Assuming you began at node a, give the order of traversal if you visited every node.

For DFS:
For BFS:

Give two different possible topological sorts of this graph: