CS61B

Lecture 27: Graphs II: Traversals

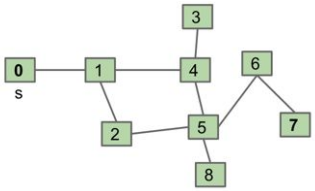
- DepthFirstPaths Implementation
- General Graph Traversals
- Topological Sorting
- Breadth First Search

Depth First Paths

Depth First Search Implementation

Common design pattern in graph algorithms: Decouple type from processing algorithm.

- Create a graph object.
- Pass the graph to a graph-processing method (or constructor) in a client class.
- Query the client class for information.



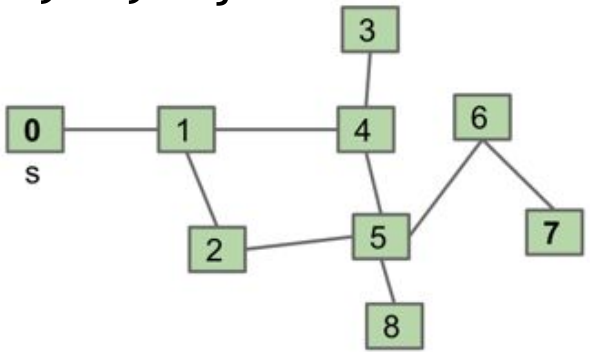
Paths.java

```
public class Paths {
    public Paths(Graph G, int s):    Find all paths from G
    boolean hasPathTo(int v):        is there a path from s to v?
    Iterable<Integer> pathTo(int v): path from s to v (if any)
}
```

Example Usage

Start by calling: `Paths P = new Paths(G, 0);`

- `P.hasPathTo(3); //returns true`
- `P.pathTo(3); //returns {0, 1, 4, 3}`



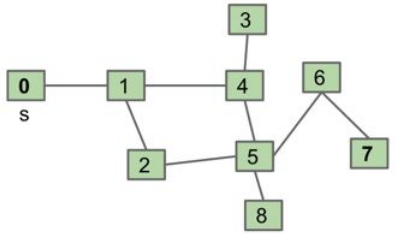
Paths.java

```
public class Paths {
    public Paths(Graph G, int s):    Find all paths from G
    boolean hasPathTo(int v):        is there a path from s to v?
    Iterable<Integer> pathTo(int v): path from s to v (if any)
}
```

Implementing Paths With Depth First Search

To visit a vertex v:

- Mark vertex v.
- Recursively visit all unmarked vertices adjacent to v.



Paths.java

Data Structures:

- `boolean[] marked`
- `int[] edgeTo`
  - `edgeTo[4] = 1`, means we went from 1 to 4.

```
public class Paths {
    public Paths(Graph G, int s):    Find all paths from G
    boolean hasPathTo(int v):        is there a path from s to v?
    Iterable<Integer> pathTo(int v): path from s to v (if any)
}
```

DepthFirstPaths and Depth First Search

Demo: DepthFirstPaths

DepthFirstPaths and Depth First Search

Demo: DepthFirstPaths

“Depth First Search” is a more general term for any graph search algorithm that traverses a graph as deep as possible before backtracking.

- The term is used for several slightly different algorithms. For example:
  - DFS may refer to the version from the previous lecture that doesn’t use any marks (and thus can get caught in a loop).
  - DFS may refer to the version where vertices are marked.
  - DFS may refer to a version where vertices are marked and source edges recorded (as in Depth First Paths).
  - DFS may refer to other algorithms like the “topological sort algorithm” well discuss later in lecture.
  - And more!

DepthFirstPaths, Recursive Implementation

```
public class DepthFirstPaths {
    private boolean[] marked;
    private int[] edgeTo;
    private int s;

    public DepthFirstPaths(Graph G, int s) {
        ...
        dfs(G, s);
    }

    private void dfs(Graph G, int v) {
        marked[v] = true;
        for (int w : G.adj(v)) {
            if (!marked[w]) {
                edgeTo[w] = v;
                dfs(G, w);
            }
        }
    }
}
```

marked[v] is true iff v connected to s

edgeTo[v] is previous vertex on path from s to v

not shown: data structure initialization

find vertices connected to s.

recursive routine does the work and stores results in an easy to query manner!

Question: How would we write hasPathTo(v)?

Graph Problems

Problem	Problem Description	Solution	Efficiency
s-t paths	Find a path from s to every reachable vertex.	DepthFirstPaths.java Demo	Θ(V+E) time Θ(V) space

Runtime is Θ(V+E)

- Each vertex is visited once and each edge is used exactly once. Each visit costs constant time.

Space is Θ(V).

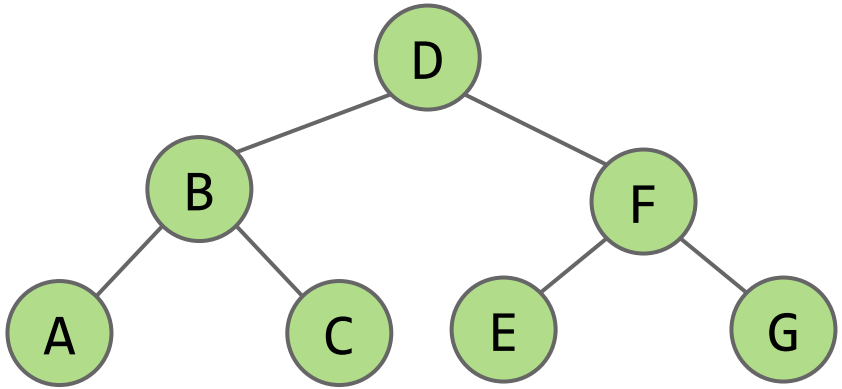
- Call stack depth is at most V.

Graph Traversals

Graph Traversals (warm up for rest of lecture)

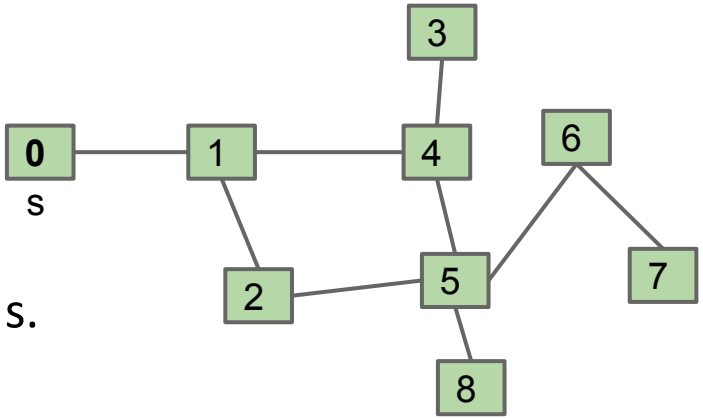
Just as there are many tree traversals:

- Preorder: DBACFEG
- Inorder: ABCDEFG
- Postorder: ACBEGFD
- Level order: DBFACEG

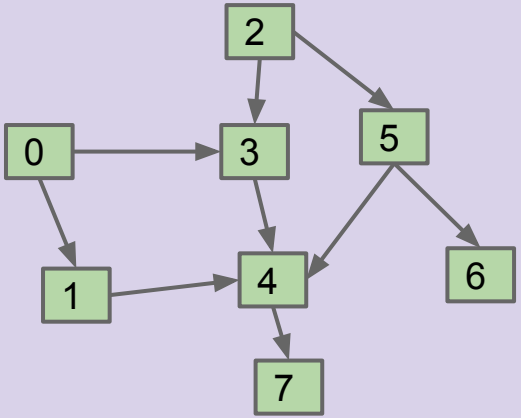


So too are there many graph traversals, given some source:

- DFS Preorder, order of DFS calls
  - 012543678
- DFS Postorder, order of returns from DFS
  - 347685210
- Level-order: Order of increasing distance from s.
  - 0 1 24 53 68 7



Level Order Traversals: http://yellkey.com/become

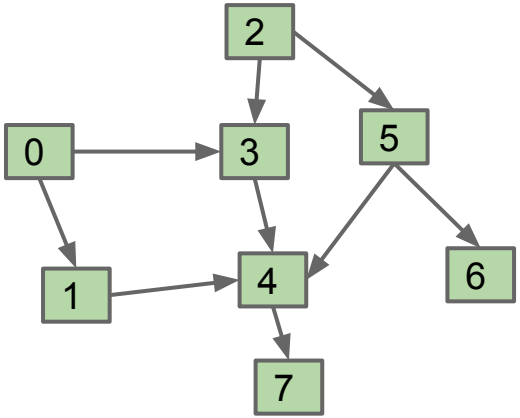


Warning: Level order is a non-standard term for graphs. The usual phrasing (as we'll learn later) is "the order visited by breadth first search".

Which of the following is **not** a valid level order traversal from vertex 2?

A. 2 3 5 4 6 7
B. 2 3 4 7 5 6
C. 2 5 3 6 4 7
D. 2 3 5 6 4 7

Level Order Traversals: http://yellkey.com/become

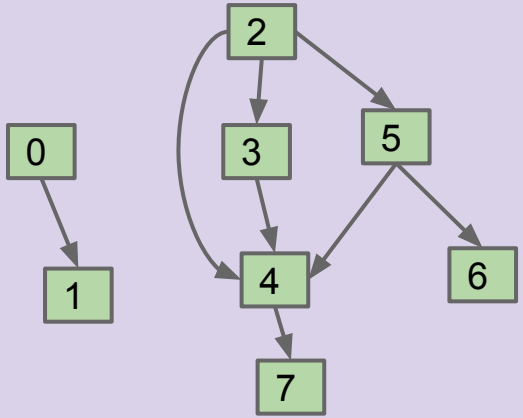


Warning: Level order is a non-standard term for graphs. The usual phrasing (as we'll learn later) is "the order visited by breadth first search".

Which of the following is **not** a valid level order traversal from vertex 2?

A. 2 3 5 4 6 7
B. 2 3 4 7 5 6
C. 2 5 3 6 4 7
D. 2 3 5 6 4 7

Level Order Traversals: http://yellkey.com/fine

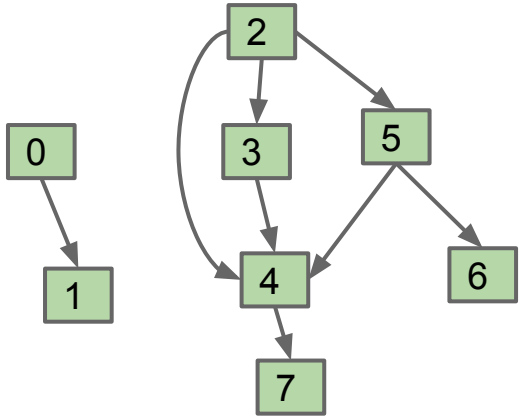


Warning: Level order is a non-standard term for graphs. The usual phrasing (as we'll learn later) is "the order visited by breadth first search".

This time, which are **valid** level order traversals from vertex 2?

A. 2 3 5 4 6 7
B. 2 3 4 7 5 6
C. 2 5 3 6 4 7
D. 2 3 5 6 4 7

Level Order Traversals: http://yellkey.com/fine

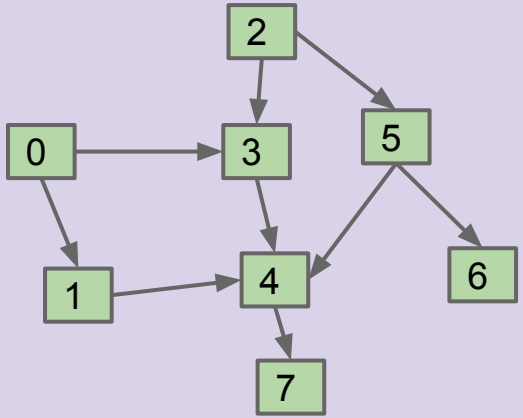


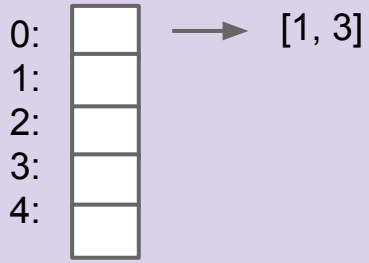
Warning: Level order is a non-standard term for graphs. The usual phrasing (as we'll learn later) is "the order visited by breadth first search".

This time, which are **valid** level order traversals from vertex 2?

A. 2 3 5 4 6 7
B. 2 3 4 7 5 6
C. 2 5 3 6 4 7
D. 2 3 5 6 4 7

DFS Postorder: http://yellkey.com/help

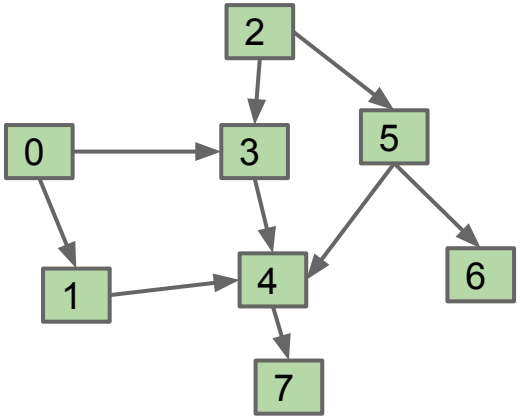


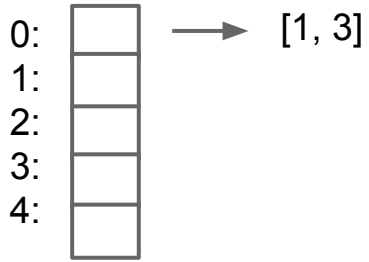


What is the DFS postorder of this graph starting at 0? Assume items appear in adjacency lists in increasing order. DFS postorder is the order of *returns* from DFS.

A. 0 1 3 4 7
B. 7 4 3 1 0
C. 7 4 1 0 3
D. 7 4 1 3 0
E. 0 1 4 7 3

DFS Postorder: http://yellkey.com/help

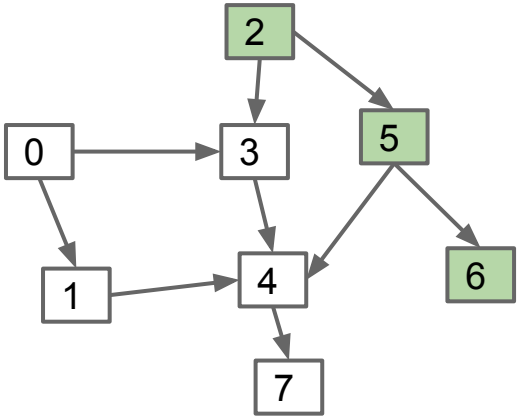


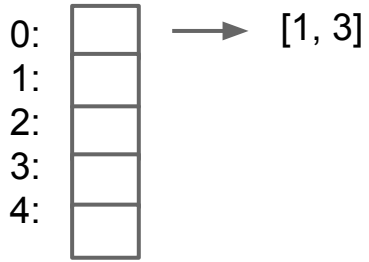


What is the DFS postorder of this graph starting at 0? Assume items appear in adjacency lists in increasing order. DFS postorder is the order of *returns* from DFS.

A. 0 1 3 4 7
B. 7 4 3 1 0
C. 7 4 1 0 3
D. 7 4 1 3 0
E. 0 1 4 7 3

DFS Postorder: http://yellkey.com/help



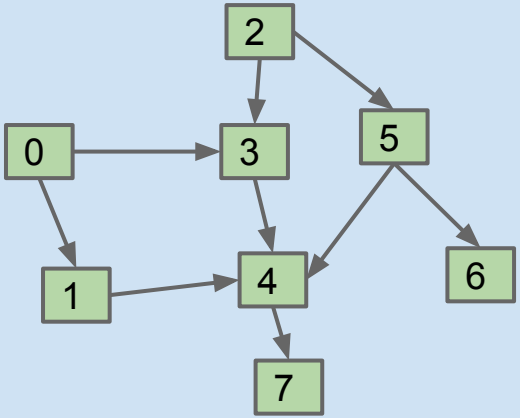


What is the DFS postorder of this graph starting at 0? Assume items appear in adjacency lists in increasing order. DFS postorder is the order of *returns* from DFS.

A. 0 1 3 4 7
B. 7 4 3 1 0
C. 7 4 1 0 3
D. 7 4 1 3 0
E. 0 1 4 7 3

Topological Sorting

Topological Sort

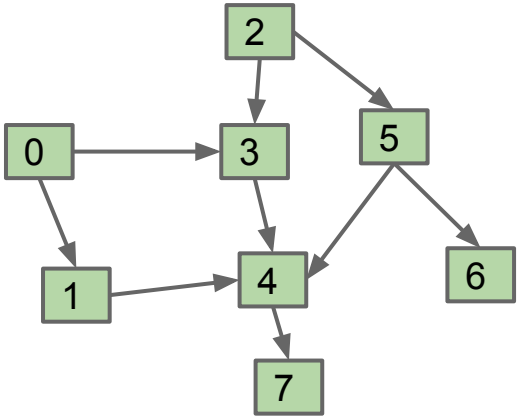


Suppose we have tasks 0 through 7, where an arrow from v to w indicates that v must happen before w.

- What algorithm do we use to find a valid ordering for these tasks?
- Valid orderings include: [0, 2, 1, 3, 5, 4, 7, 6], [2, 0, 3, 5, 1, 4, 6, 7], …

Any suggestions on where we'd start?

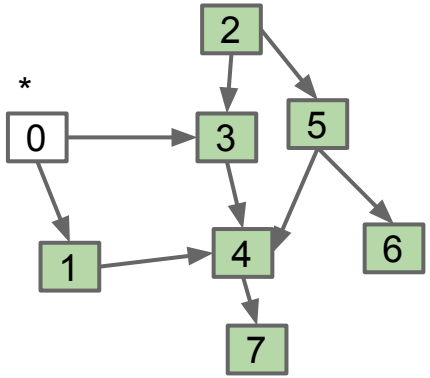
Solution (Spoiler Alert)



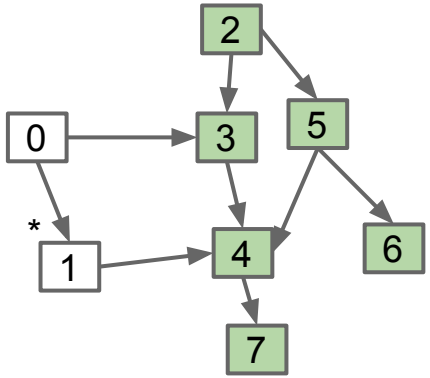
Perform a DFS traversal from every vertex with indegree 0, NOT clearing markings in between traversals.

- Record DFS post order in a list.
- Topological ordering is given by the reverse of that list (reverse postorder).

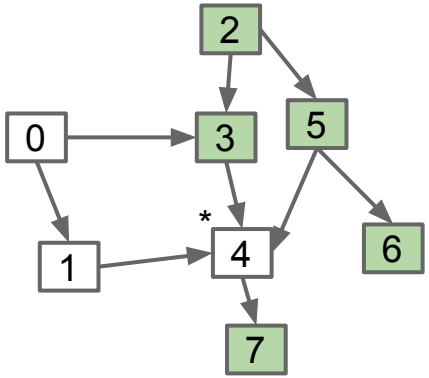
Topological Sort (Demo 1/2)



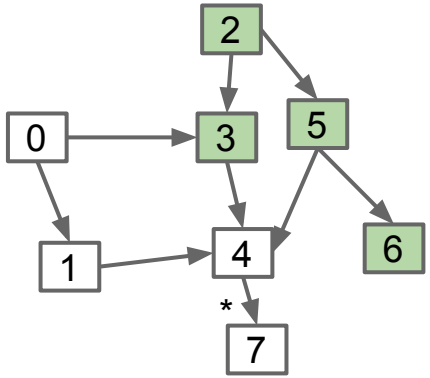
Postorder: []



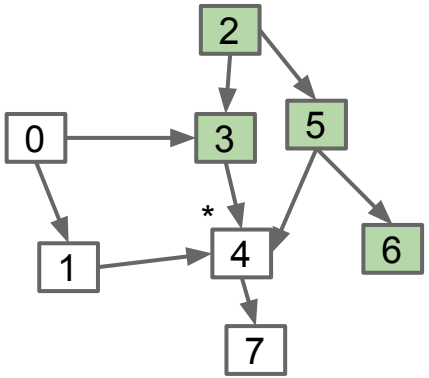
Postorder: []



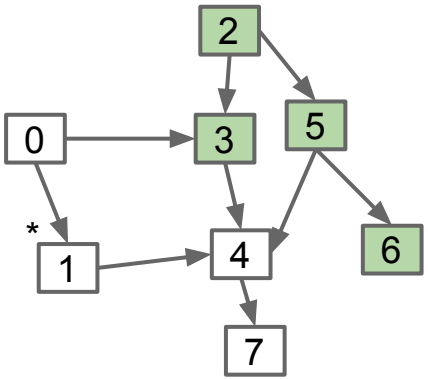
Postorder: []



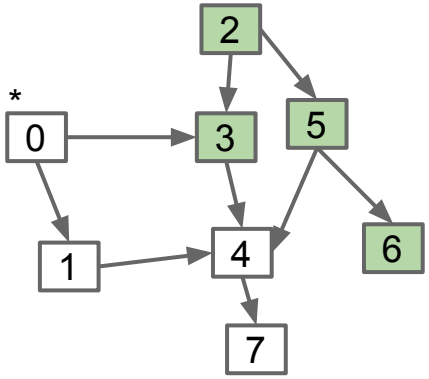
Postorder: [7]



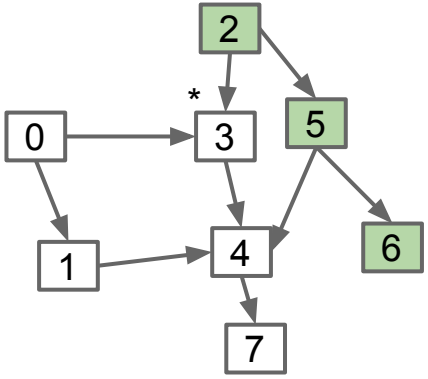
Postorder: [7, 4]



Postorder: [7, 4, 1]

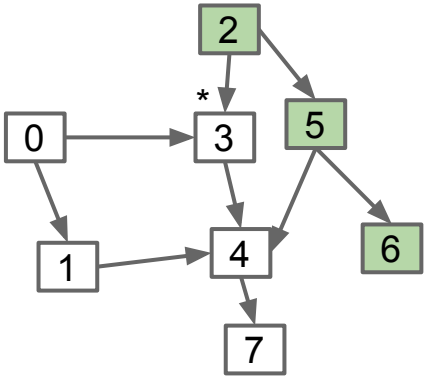


Postorder: [7, 4, 1]

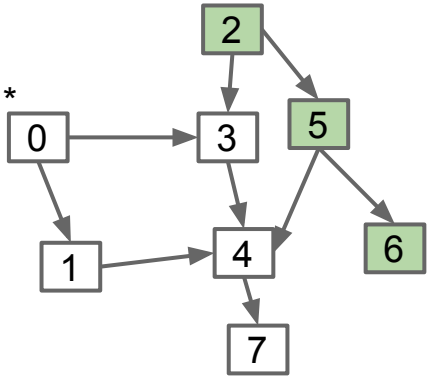


Postorder: [7, 4, 1, 3]

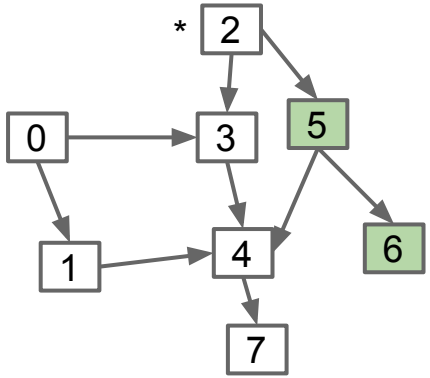
Topological Sort (Demo 2/2)



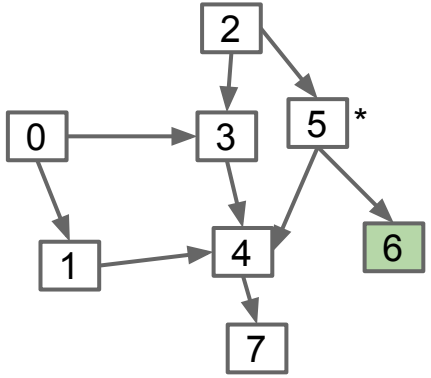
Postorder: [7, 4, 1, 3]



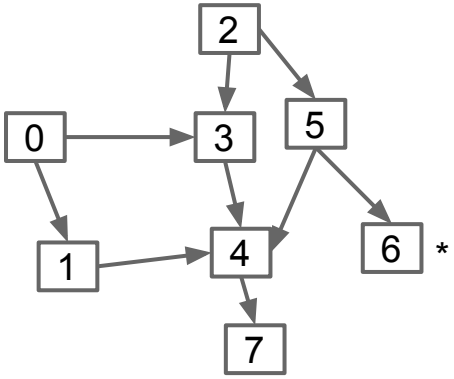
Postorder: [7, 4, 1, 3, 0]



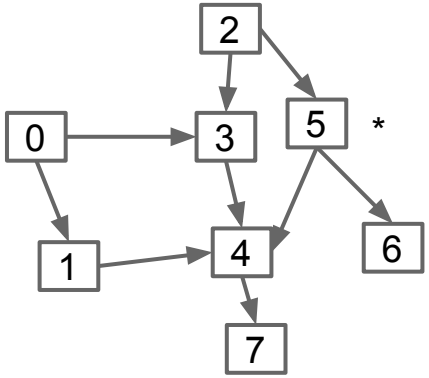
Postorder: [7, 4, 1, 3, 0]



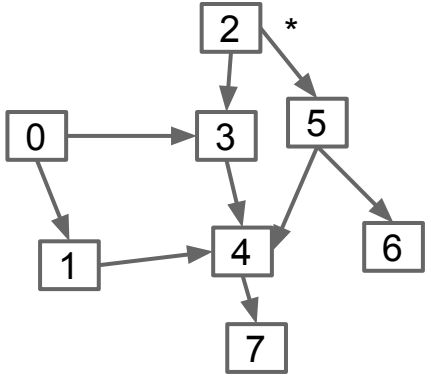
Postorder: [7, 4, 1, 3, 0]



Postorder: [7, 4, 1, 3, 0, 6]

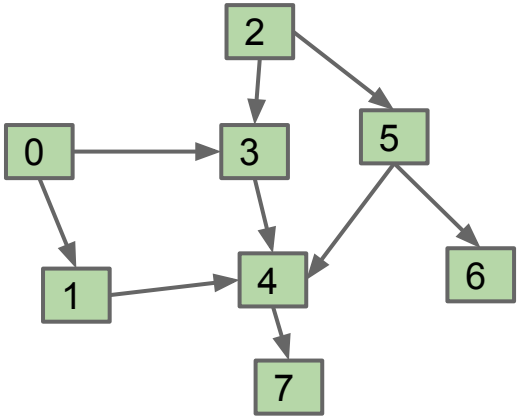


Postorder: [7, 4, 1, 3, 0, 6, 5]



Postorder: [7, 4, 1, 3, 0, 6, 5, 2]

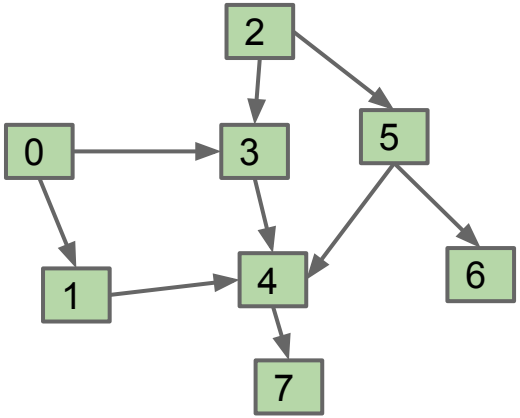
Solution (Spoiler Alert)

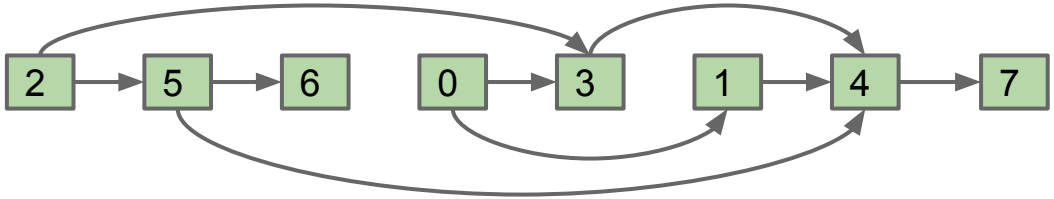


Perform a DFS traversal from every vertex with indegree 0, NOT clearing markings in between traversals.

- Record DFS post order in a list: [7, 4, 1, 3, 0, 6, 5, 2]
- Topological ordering is given by the reverse of that list (reverse postorder):
  - [2, 5, 6, 0, 3, 1, 4, 7]

Topological Sort





The reason it's called topological sort: Can think of this process as sorting our nodes so they appear in an order consistent with edges, e.g. [2, 5, 6, 0, 3, 1, 4, 7]

- When nodes are sorted in diagram, arrows all point rightwards.

Topological Sort Implementation

```
public class DepthFirstOrder {
    private boolean[] marked;
    private Stack<Integer> reversePostorder;
    public DepthFirstOrder(Digraph G) {
     reversePostorder = new Stack<Integer>();
     marked = new boolean[G.V()];
    for (int v = 0; v < G.V(); v++) {
        if (!marked[v]) { dfs(G, v); }
    }
    private void dfs(Digraph G, int v) {
        marked[v] = true;
        for (int w : G.adj(v)) {
            if (!marked[w]) { dfs(G, w); }
         }
         reversePostorder.push(v);
    }
    public Iterable<Integer> reversePostorder()
    { return reversePostorder; }
}
```

Textbook implementation (shown here) uses a stack instead of a creating a list and then reversing it.

Create empty stack.

Perform DFS of all unmarked vertices. Note: Our algorithm earlier started at vertices with indegree zero. It turns out this algorithm works no matter where you start!

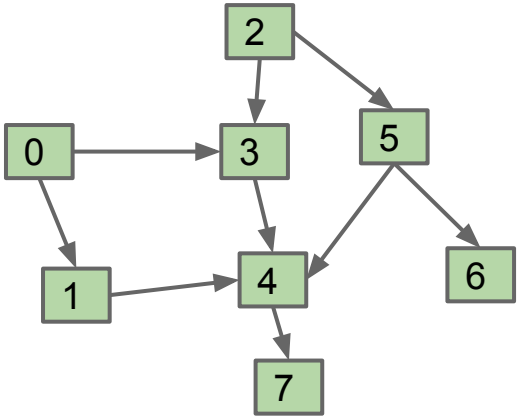
After each DFS is done, ‘visit’ vertex by putting on a stack.

Graph Problems

Problem	Problem Description	Solution	Efficiency
s-t paths	Find a path from s to every reachable vertex.	DepthFirstPaths.java Demo	Θ(V+E) time Θ(V) space
topological sort	Find an ordering of vertices consistent with directed edges.	DepthFirstOrder.java Demo	Θ(V+E) time Θ(V) space

Breadth First Search

Breadth First Search



Goal: Given the graph above, find the shortest path from 0 to every other vertex.

- Level-order provides the shortest paths from 0 to every reachable vertex from 0 (by definition!)

Finding the “Level-Order”

For trees (a special type of graph), we used iterative deepening.

- Bad for spindly trees, $\Theta(N^2)$.

Breadth First Search.

- Initialize a queue with a starting vertex s and mark that vertex.
  - Let’s call this the ***fringe***.
- Repeat until queue is empty:
  - Remove vertex v from queue.
  - Add to the queue any unmarked vertices adjacent to v and mark them.

BreadthFirstPaths demo.

From here on out, we’ll refer to “level order” as “the order visited by BFS”.

BreadthFirstPaths Implementation

```
public class BreadthFirstPaths {
  private boolean[] marked;
  private int[] edgeTo;
  ...

  private void bfs(Graph G, int s) {
  Queue<Integer> fringe =
          new Queue<Integer>();
  fringe.enqueue(s);
  marked[s] = true;
  while (!fringe.isEmpty()) {
    int v = fringe.dequeue();
    for (int w : G.adj(v)) {
      if (!marked[w]) {
        fringe.enqueue(w);
        marked[w] = true;
        edgeTo[w] = v;
      }
    }
  }
}
```

marked[v] is true iff v connected to s

edgeTo[v] is previous vertex on path from s to v

set up starting vertex

for freshly dequeued vertex v, for each neighbor that is unmarked:

- Enqueue that neighbor to the fringe.
- Mark it.
- Set its edgeTo to v.

Breadth First Search

Our code for BFS examines vertices in increasing order from source.

- Why? The fringe queue always consists of the following (for some k):
  - ≥ 0 vertices of distance k from s,
  - ≥ 0 vertices of distance k + 1 from s
- Handy for finding shortest paths.

Runs in V+E time and uses V space:

- Each vertex and each edge processed a constant number of times.
- No vertex may be enqueued more than once.

BreadthFirstSearch for Google Maps

Would breadth first search be a good algorithm for a navigation tool (e.g. Google Maps)?

- Assume vertices are intersection and edges are roads connecting intersections.

Some roads are longer than others.

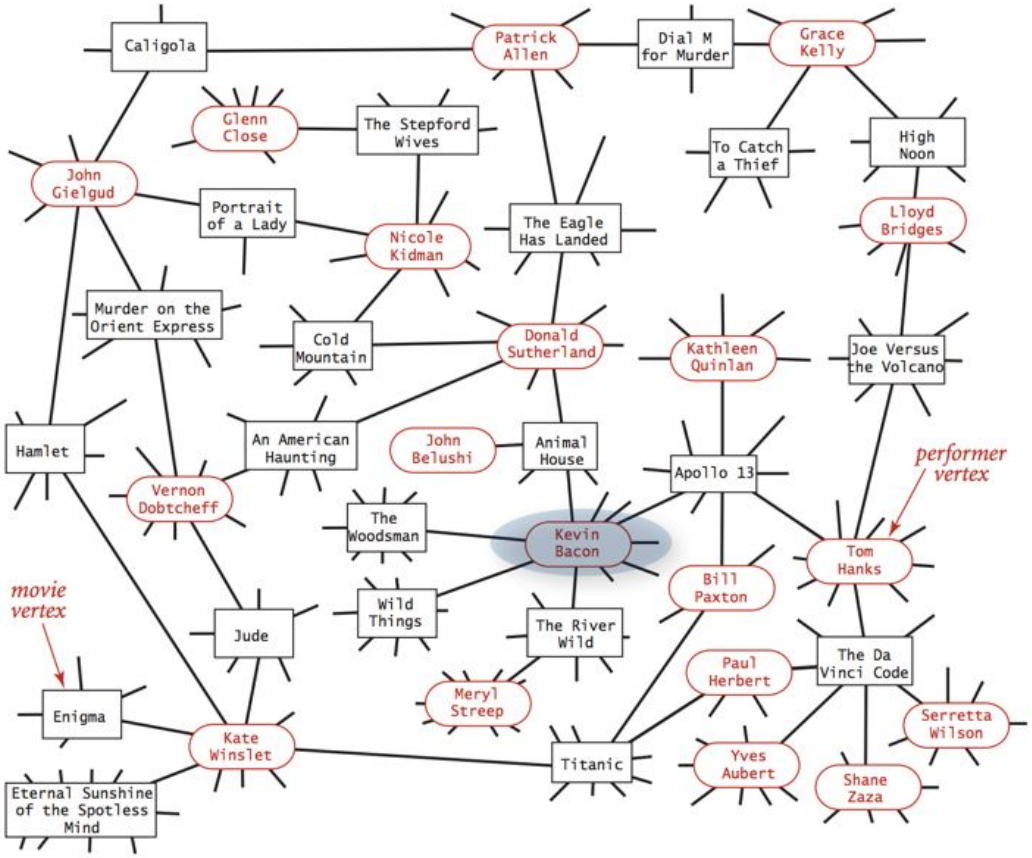
- BAD!

One use of BFS: Kevin Bacon

Graph with two types of vertices:

- Movies
- Actors

Perform BFS from s=Kevin Bacon.



Citations

http://www.gosidemount.com/Guided_Diving/images/guided_cavern.jpg