Topological Sort

• Any partial order can be represented by a directed acyclic graph (DAG) $G = (V,E)$

• A topological sort is an ordering of all of $G$'s vertices $v_1, v_2, \ldots, v_n$ such that...

**Before:** for every edge $(v_i, v_k)$ in $E$, $i<k$.

**After:** list all nodes of $G$ such that all arrows are pointing to the right.

Topological sort

• There are often many possible topological sorts of a given DAG

• Topological orders for this DAG:

  • 1, 2, 5, 4, 3, 6, 7
  • 2, 1, 5, 4, 7, 3, 6
  • 2, 5, 1, 4, 7, 3, 6
  • Etc.

• Each topological order is a *feasible schedule.*
Topological Sorts for Cyclic Graphs?

Impossible!

• If v and w are two vertices on a cycle, there exist paths from v to w and from w to v.
• Any ordering will contradict one of these paths

Topological sort algorithm

• Algorithm
  – Assume indegree is stored with each node.
  – Repeat until no nodes remain:
    • Choose a node of zero indegree and output it.
    • Remove the node and all its edges and update indegres.

Example

• Indegree
  • 0: 1, 2
  • 1: 5
  • 2: 3, 6, 7
  • 3: 4
• Pick 1
Example

- Indegree
- 0: 2
- 1: 5, 3
- 2: 6, 7, 4
- 3:
  - Pick 2

Example

- Indegree
- 0: 5
- 1: 3, 4
- 2: 6, 7
- 3:
  - Pick 5

Example

- Indegree
- 0: 4
- 1: 3, 7
- 2: 3, 6
- 3:
  - Pick 4
Example

• Indegree
• 0: 3, 7
• 1: 6
• 2:
• 3:

• Pick 3

Example

• Indegree
• 0: 7, 6
• 1: 6
• 2:
• 3:

• Pick 6

Example

• Indegree
• 0: 7, 6
• 1: 6
• 2:
• 3:

• Pick 7
**Graph Traversals**

**Use of a stack**

- It is very common to use a stack to keep track of:
  - nodes to be visited next, or
  - nodes that we have already visited.
- Typically, use of a stack leads to a *depth-first* visit order.
- Depth-first visit order is “aggressive” in the sense that it examines complete paths.
Topological Sort as DFS

- do a DFS of graph G
- as each vertex v is “finished” (all of it’s children processed, i.e., postorder in the search tree), insert it onto the front of a linked list
- return the linked list of vertices

why is this correct?

Use of a queue

- It is very common to use a queue to keep track of:
  - nodes to be visited next, or
  - nodes that we have already visited.
- Typically, use of a queue leads to a breadth-first visit order.
- Breadth-first visit order is “cautious” in the sense that it examines every path of length i before going on to paths of length i+1.

Graph Searching ???

- Graph as state space (node = state, edge = action)
- For example, game trees, mazes, ...
- BFS and DFS each search the state space for a best move. If the search is exhaustive they will find the same solution, but if there is a time limit and the search space is large...
- DFS explores a few possible moves, looking at the effects far in the future
- BFS explores many solutions but only sees effects in the near future (often finds shorter solutions)