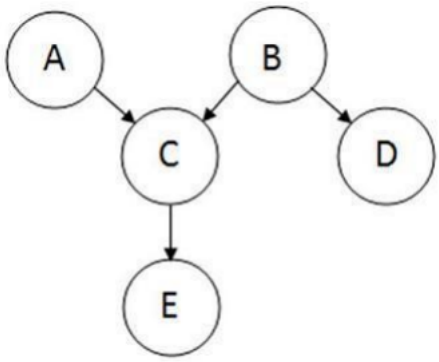
Topological Sort

1. Idea: Given a DAG, output all vertices in an order so that no vertex appears before another vertex that points to it.

2. Algorithm Idea:
   Keep track of the in-degree of each node.
   Use a queue to ensure the proper ordering of nodes (from least to greatest in-degree)
   Every time an in-degree is 0, enqueue it.
   Every time a node is processed, decrement its adjacents in-degree.

3. Example:



4. Running time:
   Initialization: $O(|V| + |E|)$ (assuming adjacency list)
   Sum of all enqueues and dequeues: $O(|V|)$
   Sum of all decrements: $O(|E|)$ (assuming adjacency list)

   So total is $O(|E| + |V|)$ - much better for sparse graphs

Graph Traversals

1. Depth-First Search:
   Recursively explore one part before going back to the other parts not yet explored
   Typically use a stack to keep track of which nodes to process next (non-recursive)

2. DFS(Node start) {
   mark and process start;
   for each node u adjacent to start
   if u is not marked
   DFS(u)
   }

3. Breadth-First Search:
   explore areas closer to the start node first
   Typically use a queue to keep track of which nodes to process next

4. BFS(Node start) {
   initialize queue q and enqueue start;
   mark start as visited
   while(q is not empty) {
   next = q.dequeue() // and process
   for each node u adjacent to next
   if(u is not marked)
   mark u and enqueue onto q } }

5. Comparison:
   Breadth-first finds shortest paths
   Better for what is the shortest path from x to y

   But depth-first can use less space in finding a path

   A third approach:
   Iterative deepening (IDFS):
   Try DFS but disallow recursion more than K levels deep
   If that fails, increment K and start the entire search over
   Like BFS, finds shortest paths. Like DFS, less space.