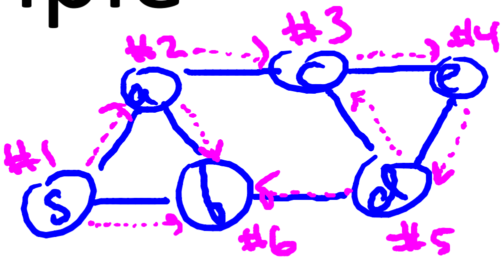
Graph Primitives

Depth-First Search

Design and Analysis of Algorithms I

Overview and Example



Depth-First Search (DFS) : explore aggressively, only backtrack when necessary.

-- also computes a topological ordering of a directed acyclic graph
-- and strongly connected components of directed graphs

Run Time : O(m+n)

The Code

Exercise : mimic BFS code, use a stack instead of a queue [+ some other minor modifications]

Recursive version : DFS(graph G, start vertex s)

-- mark s as explored

-- for every edge (s,v) :

  -- if v unexplored

    -- DFS(G,v)

Basic DFS Properties

Claim #1 : at the end of the algorithm, v marked as explored <==> there exists a path from s to v in G.

Reason : particular instantiation of generic search procedure

Claim #2 : running time is $O(n_s+m_s)$,

where n_s = # of nodes reachable from s

m_s = # of edges reachable from s

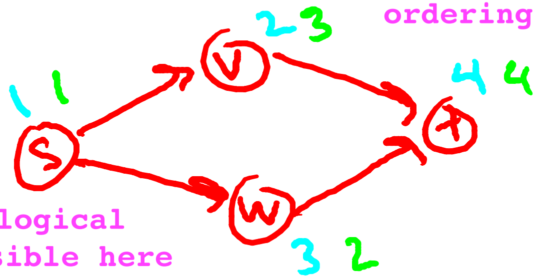
Reason : looks at each node in the connected component of s at most once, each edge at most twice. same as BFS

Application: Topological Sort

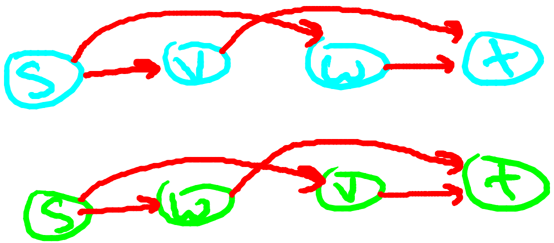
can have > 1 topological ordering

Definition : A topological ordering of a directed graph G is a labeling f of G's nodes such that:

1. The f(v)'s are the set {1,2,..,n}
2. $(u,v) \in G => f(u) < f(v)$



only 2 topological orderings possible here



Motivation : sequence tasks while respecting all precedence constraints.

Note : G has directed cycle => no topological ordering

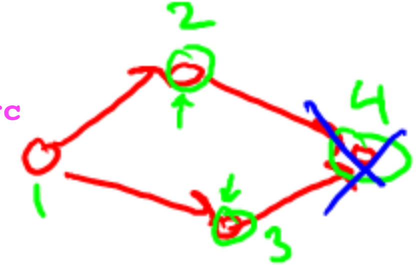
Theorem : no directed cycle => can compute topological ordering in O(m+n) time.

(non DFS solution. good implementation of this is linear time. but DFS solution is neat)

Straightforward Solution

Note : every directed acyclic graph has a sink vertex.

vertex with no outgoing arc



Reason : if not, can keep following outgoing arcs to produce a directed cycle.

To compute topological ordering :

-- let v be a sink vertex of G pick arbitrarily if > 1 sink vertex

-- set f(v) = n delete incoming edges to v

-- recurse on G-{v}

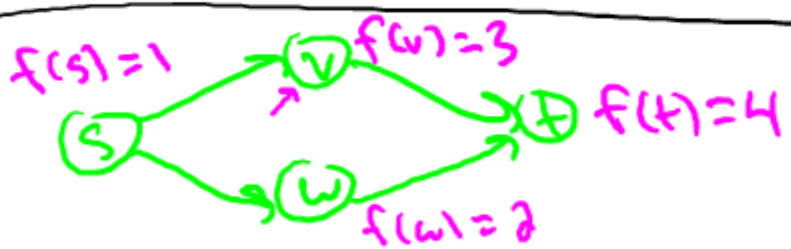
Why does it work? : when v is assigned to position i, all outgoing arcs already deleted => all lead to later vertices in ordering.

embed the DFS in outer for loop just like you did for SCC with BFS.
makes sure we can invoke DFS multiple times if we have > 1 connected components

Topological Sort via DFS (Slick)

DFS-Loop (graph G)
-- mark all nodes unexplored
-- current-label = n [to keep track of ordering]
-- for each vertex
  -- if v not yet explored [in previous DFS call]
    -- DFS(G,v)

DFS(graph G, start vertex s)
-- for every edge (s,v)
  -- if v not yet explored
    -- mark v explored
    -- DFS(G,v)
-- set f(s) = current_label
-- current_label = current_label-1



Topological Sort via DFS (con’d)

Running Time : O(m+n).

Reason : O(1) time per node, O(1) time per edge.

Correctness : need to show that if (u,v) is an edge, then f(u) < f(v)

(since no directed cycles)

Case 1 : u visited by DFS before v => recursive call corresponding to v finishes before that of u (since DFS).
$\Rightarrow$f(v) > f(u)

Case 2 : v visited before u => v’s recursive call finishes before u’s even starts. => f(v) > f(u)

Q.E.D.