Topological Sort

Directed acyclic graph (DAG)

- Directed acyclic graph (DAG) gives a partial ordering
- If \( G \) contains edge \((u, v)\) then \( u \) appears before \( v \) in ordering
- Linear ordering of vertices

Topological sort of \( G \):

Can view edges as ordering constraints

Directed acyclic graph (DAG)

Overview

Strongly connected components

\[ \text{Topological Sort} \]
TOPOLOGICAL-SORT(G)
1 call DFS(G) to compute finishing times f_v for each vertex v
2 as each vertex is finished, insert it onto the front of a linked list
3 return the linked list of vertices

Why not discovery time?
Why is that important?
Nodes ordered by inverse of finish times

Code

Example

How do we turn this into a topological sort?

Depth-First Search with discover/finish times

• Nodes ordered by inverse of finish times
Topological Sort

- Correctness?
- Time Complexity?
- Lemma 22.11

A directed graph G is acyclic if a depth-first search of G yields

G is acyclic ⇒ no back edges

G is acyclic ⇐ no back edges

A directed graph G is acyclic iff a depth-first search of G yields no back edges.

Discovery and Finish Time

- If u is an ancestor of v - u.f > v.f and u.d < v.d
- So far, seems ordering by discovery time or inverse finish seem the same
  - If n is an ancestor of a
  - p.n > p.n and f.a < f.n
  - So far, seems ordering by discovery time or inverse finish seem the same
  - Ordering by reverse finish time puts a before n
  - Ordering by discovery time puts n before a wrong
  - A cross edge:
  - Not explored earlier since goes from a to n, not n to a
  - But, there might be an edge/path from a to n
  - If n is discovered first: u.d < v.d

+ If and a not ancestor/descendant
Theorem 22.12

Topological sort produces a topological sort of the DAG

Proof by contradiction

Assume $u.f < v.f$

By Parenthesis theorem, 3 cases exist, two of which are consistent with $u.f < v.f$

Case 1: $u$ is an descendant of $v$

Then $(u, v)$ is a back edge, so has a cycle

Contraction

Case 2: $u.f < v.d$

DFS would have discovered $v$ in searching through $u$'s edges

Contradiction

Need to prove that if $G$ has cycle $(a, n)$, then $f.n > f.a$

Topological sort produces a topological sort of the DAG

Theorem 22.12

Proof

$G$ is acyclic $\Rightarrow$ no back edge

While path $a, n$ becomes descendant of $a$ in DFS forest $n$

Among vertices of form a path of white vertices from $a$ to $n$

Let c be the proceeding edge in c

Let a be the first vertex in c to be discovered

Assume there is a cycle c

Therefore there is a back edge

Thus there is a path from a to n in c, and (a, n) completes cycle

$G$ is acyclic $\Rightarrow$ no back edge

$G$ is acyclic $\Leftrightarrow$ no back edge
Strongly Connected Components

- **Definition**: Directed graph where every two vertices are mutually reachable.
- **Properties**:
  - Reflexive: \( u \in C \) then \( u \leftrightarrow u \)
  - Symmetric: \( u \leftrightarrow v \) then \( v \leftrightarrow u \)
  - Transitive: \( u \leftrightarrow v \) and \( v \leftrightarrow w \) then \( u \leftrightarrow w \)

Strongly connected components can define an equivalence class:

- Maximal set \( C \subseteq V \) such \( u \leftrightarrow v \) for all \( u, v \in C \)
- Partitions the vertices into distinct sets

Classic application of DFS
- Decomposing DAG into its strongly connected components
- Useful as the basis of a lot of graph algorithms

**Overview**
- Topological Sort
- Strongly Connected Components
DFS and SCC

- Starting DFS in a node in a SCC. Will it find all of the vertices in C?
- Will it find other vertices?
- What if we reverse the graph \( G_T \)? Flip all of the edges.
- Will it find other vertices?
- Will it find all of the vertices in C?

Questions

- Can a DAG have any non-trivial strongly connected components?
- Does it make sense to talk about strongly connected components for undirected graphs?
- Does it have the same SCCs?
- What if we reverse the edges of a graph \( G \) to form \( G_T \)?
- Does it have the same SCCs?
Example

Happened to start at $c$, then $b$

- Component graph
  + Dark nodes are root nodes
    - By decreasing $u.f$
    - Main loop: order vertices
    + Reverse edges
    + Second pass
  + Happened to start at $c$, then $b$

- First pass

Code

```plaintext
1 call DFS($G$) to compute finishing times $u.f$ for each vertex
2 compute $G_T$
3 call DFS($G_T$), but in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component

The idea behind this algorithm comes from a key property of the component $G_T$ is the $G$ but with the edges switched

- Note: SCC are preserved under graph transpose
- Time Complexity?

STROMOY-CONNECTED-COMPONENTS $(G)$

1 call DFS($G$) to compute finishing times $f.n$ for each vertex $n$
2 compute $G_T$
3 call DFS($G_T$), but in the main loop of DFS, consider the vertices in order of decreasing $f.n$
4 output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component
```

Note: SCC are preserved under graph transpose

$G_T$ is the $G$ but with the edges switched

Time Complexity?
Lemma 22.13

Let $C$ and $C'$ be distinct SCCs in DAG $G = (V, E)$. Let $u, v \in C$, and $u', v' \in C'$. Suppose $G$ has path $u \leadsto v$, then $G$ cannot have path $v' \leadsto v$.

Proof: If $G$ has path $v' \leadsto v$, then $G$ contains path $u \leadsto v' \leadsto v$. So $u$ and $v'$ are mutually readable so must be in the same component. Contradiction.

Towards proving algorithm is correct

**Component Graph**

- $G_{SCC} = (V_{SCC}, E_{SCC})$
  - $V_{SCC}$ contains one vertex $v_i$ for each SCC $C_i$.
  - $(v_i, v_j) \in E_{SCC}$ if $G$ contains a directed edge $(x', y')$ for some $x \in C_i$ and $y \in C_j$.

- Graph is same as contracting all edges whose vertices are in the same SCC.

- Prove that the $G_{SCC}$ is a DAG.

- $G_{SCC}$ contains one vertex $v_i$ for each SCC $C_i$.

- $G_{SCC}$ contains vertex $u_i$ for each vertex $u$.

- $G_{SCC}$ contains one vertex $v_i$ for each SCC $C_i$.

- Towards proving algorithm is correct.