Overview

⇒ Topological Sort

• Strongly Connected Components

Topological Sort

• Only for Directed acyclic graph (DAG)
• Can view edges as ordering constraints
  - Edge \((u,v)\) means \(u\) comes before \(v\)
  - Consistent since no cycles
• Topological sort of \(G = (V,E)\):
  - Linear ordering of vertices
    if \(G\) contains edge \((u,v)\), then \(u\) appears before \(v\) in ordering
• Graph might just give a partial ordering
  - If \(G\) has edges \((u,v)\) and \((u,w)\), \(u\) is first, but what is 2nd?
  - Can be many different linear orderings

*Why must graph be directed and acyclic?
Example

- Depth-First Search with discover/finish times

```
11/16  undershorts
       ↓
12/15  pants
       ↓
       ↓
       6/7  belt
       ↓
       ↓
       2/5  tie
       ↓
       ↓
       3/4  jacket

17/18  socks
       ↓
       ↓
       ↓
       ↓
       ↓
       9/10  watch
       ↓
       ↓
       ↓
       ↓
       ↓
       13/14  shoes
```

- How do we turn this into a topological sort?

Increasing Discovery Times?

```
11/16  undershorts
       ↓
12/15  pants
       ↓
       ↓
       6/7  belt
       ↓
       ↓
       2/5  tie
       ↓
       ↓
       3/4  jacket

17/18  socks
       ↓
       ↓
       ↓
       ↓
       ↓
       9/10  watch
       ↓
       ↓
       ↓
       ↓
       ↓
       13/14  shoes
```

- Sort by discovery times?
  - Shirt, tie, jacket, so far so good
  - belt, watch, undershorts, pants, oops
  - shoes, socks, oops

*Why did it break?
Example Continued

- By Parenthesis Theorem, either
  - \( u.d < v.d \) and \( v.f < u.f \)
    + \( v \) is a descendent of \( u \)
    + \( u \) must come before \( v \) in top. sort
  - \( v.f < u.d \ u.f \)
    + There is no path from \( v \) to \( u \)
    + But there could be path from \( u \) to \( v \)
    + \( u \) should come before \( v \) in top. sort

- In both cases, if \( v.f < u.f \), \( u \) should come before \( v \)
  - Sort by decreasing values of finish time

Code

TOPOLOGICAL-SORT(G)

```plaintext
1 call DFS(G) to compute finishing times \( v.f \) 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
```

- Nodes ordered by inverse of finish times
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 discover time or inverse finish seem the same

- If $u$ and $v$ not ancestor/descendant
  - If $u$ is discovered first: $u.d < v.d$ and $u.f < v.f$
  - But, there might be an edge/path from $v$ to $u$
    + Not explored earlier since goes from $v$ to $u$, not $u$ to $v$
    + A cross edge!
  - Ordering by discovery time puts $u$ before $v$: wrong
  - Ordering by reverse finish time puts $v$ before $u$

Topological Sort

- Time complexity?
- Correctness?

**Lemma 22.11**
A directed graph $G$ is acyclic iff a depth-first search of $G$ yields no back edges

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

$G$ is acyclic $\Leftarrow$ no back edge
Proof

G is acyclic ⇒ no back edge
Assume it has a back-edge \((u, v)\)
So \(u\) is a descendant of \(v\) in the depth-first forest
Thus there is a path from \(v\) to \(u\) in \(G\), and \((u, v)\) completes cycle

No back edge ⇒ G is acyclic
Assume there is a cycle \(c\)
Let \(v\) be the first vertex in \(c\) to be discovered in a DFS
Let \((u, v)\) be the preceding edge in \(c\)
At time \(v.d\), vertices of \(c\) form a path of white vertices from \(v\) to \(u\)
White path theorem: \(u\) becomes descendant of \(v\) in DFS forest
Therefore \((u,v)\) is a back edge

Theorem 22.12

Topological sort produces a topological sort of the DAG
Need to prove that if \(G\) has edge \((u, v)\), then \(v.f < u.f\)
Proof by contradiction
Assume \(G\) has edge \((u, v)\) where \(u.f < v.f\) in a DFS
By Parenthesis theorem, 3 cases exists,
  two of which are consistent with \(u.f < v.f\)
Case 1: \(u\) is an descendant of \(v\)
  Then \((u, v)\) is a backedge, so has a cycle
  Contraction
Case 2: \(u.f < v.d\)
  DFS would have discovered \(v\) in searching through \(u\)’s edges
  Contradiction
Overview

• Topological Sort
  ⇒ Strongly Connected Components

Strongly Connected Components

• \( u, v \in G \) are Mutually Reachable (Appendix B.4)
  - if \( u \rightarrow v \) and \( v \rightarrow u \)
  - Properties:
    + Reflexive: \( u, u \) is mutually reachable
    + Symmetric: \( u, v \) is mutually reachable if \( v, u \) is
    + Transitive: if \( u, v \) and \( v, w \) are mutually reachable so is \( u, w \)
  - Mutually reachable can define an equivalence class

• Strongly connected components:
  - Maximal set \( C \subseteq V \) such if \( u, v \in C \) then \( u, v \) are mutually reachable
  - Partitions the vertices into distinct sets

• Classic application of DFS
  - Decomposing DAG into its strongly connected components
  - Strongly Connected Components useful as the basis of a lot of graph algorithms
Questions

- Can a DAG have any non-trivial Strongly Connected Components?

- Does it make sense to talk about Strongly Connected Components for undirected graphs?

- What if we reverse the edges of a graph $G$ to form $G^T$?
  - Does it have same SCCs?

DFS and SCC

- Starting DFS in a node in a SCC $C$:
  - 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?
  - Start it from the last finish times?
**Code**

**STRONGLY-CONNECTED-COMPONENTS(G)**

1. call DFS(G) to compute finishing times \( u.f \) for each vertex \( u \)
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

- \( G^T \) is the \( G \) but with the edges switched
  - Note: SCC are preserved under graph transpose
- Time Complexity?

---

**Example**

- **First Pass**
  + Happened to start at \( c \), then \( b \)

- **Second pass**
  + Reverse edges
  + Main loop: order vertices by decreasing \( u.f \)
  + Dark nodes are root nodes

- **Component Graph**
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 $u \rightsquigarrow u'$, then $G$ cannot have $v' \rightsquigarrow v$

**Proof:**

If it does, then $G$ contains path $u \rightsquigarrow u' \rightsquigarrow v'$ and $v' \rightsquigarrow v \rightsquigarrow u$.

So $u$ and $v'$ are mutually readable.

So must be in same component.

Contradiction
Correctness Proof of Algorithm