DFS on directed graphs

- Interpret “w adjacent to v” as finding directed edge (v, w)
- Edges (v, w) grouped into four types:
  - Tree edge
    - White w discovered from gray v
    - Actually a set of trees, or forest
  - Back edges
    - w ancestor of v or on stack when w visited (w gray)
  - Forward edges
    - w descendant of v (w black, d[v] < d[w])
  - Cross-edges
    - All others (w black, d[v] > d[w])
Topological sort

- A linear ordering of vertices of a Directed Acyclic Graph (DAG)
- For any directed edge \((u, v)\), \(u\) precedes \(v\) in ordering
Use of topological sort

- Application: nodes are tasks, edges are “precedences” (e.g. one task must be done before another can be started)
- A topological sort gives an order in which to do tasks
- Naive algorithm: look for a source (no incoming edges), choose and delete it
- This is $\Theta(n(n + m))$
Using DFS

- The finishing times \( f[u] \) give a topological ordering (taken in decreasing order).
- Equivalently, in postprocessing (when vertex coloured black), put it on front of a linked list; resulting list is topologically ordered.
- Why does this work? Intuitively OK
- Need to show that for any directed edge \((u, v)\), \( f[u] > f[v] \); this is not obvious.
Proof of topological sort

**Lemma**

A graph is acyclic iff there are no back edges in a DFS of the graph.

Proof: \((\Rightarrow)\) If there is a back edge, that edge plus the tree path forward gives a cycle.
Proof of topological sort

(⇐) If there is a cycle, let \( u \) be the first discovered cycle vertex in DFS, and let \((v, u)\) be a cycle edge.

The white-path theorem applied to \(v, u\) says that \(v\) is a descendant of \(u\), so \((v, u)\) is a back edge.
Proof of topological sort

Apply DFS to a DAG, and consider directed edge \((u, v)\); must show \(f[v] < f[u]\).

- When \((u, v)\) explored, \(v\) can not be gray, because \((u, v)\) would be a back edge.
- If \(v\) is white, it becomes descendant of \(u\), so \(f[v] < f[u]\) by parenthesis theorem.
- If \(v\) is black, \(f[v]\) already set; \(f[u]\) must be bigger when it is set.
A strongly connected component is a maximal set of vertices $C \subseteq V$ such that for any $u, v$ in $C$, there are directed paths from one to the other.
A naive algorithm for SCC

- Run DFS-visit from each node $u$ to get $reach(u) =$ vertices reachable from $u$.
- $S \leftarrow reach(u)$; for every $v$ in $S$, if $u \notin reach(v)$, delete $v$ from $S$.
- What is left is a strongly connected component.
- This takes $\Theta(n(n + m))$ time just to get one strongly connected component.
Better use of DFS for SCC

- Let \( G^T \) be \( G \) with all edges reversed.
- \( G \) and \( G^T \) have the same strongly connected components.
- Can create \( G^T \) in \( O(n + m) \) time.

Strongly-Connected-Components(\( G \))

1. Call a DFS on \( G \), recording finishing times.
2. Compute \( G^T \).
3. Call a DFS on \( G^T \), choosing roots in order of decreasing finishing time in first DFS (step 1).
4. Vertices of each tree in the depth-first forest is a strongly connected component.
SCC algorithm example

![Graph Diagram](image)

- 8/11
- 7/12
- 9/10
- 1/6
- 4/5
- 2/3
Intuition: the component graph

- Define a graph $G^{SCC}$: Each vertex is a strongly connected component of $G$.
- $(u, v)$ is an edge in $G^{SCC}$ iff there is an edge in $G$ from a vertex in the component $u$ to the component $v$. 

![Diagram of a strongly connected component graph]
The component graph

- $G^{SCC}$ is a directed acyclic graph (DAG).
- The second DFS on $G^T$ basically visits the vertices of $(G^T)^{SCC}$ in reverse topological order (or of $G^{SCC}$ in topological order).
Proof of SCC algorithm

Extend definition of $d$ and $f$ (discovery time and finishing times) to sets:
For $U \subseteq V$, $d(U) = \min_{u \in U} d[u]$ and $f(U) = \max_{u \in U} f[u]$.

Lemma 4

For two components $C$ and $C'$, if there is an edge from $C$ to $C'$, then $f(C) > f(C')$.

Proof:
If $d(C) < d(C')$, then when the first vertex $x$ was discovered in $C$, there was a white path from $x$ to all vertices in $C$ and $C'$; the white-path and parenthesis theorems show $f[x] = f(C) > f(C')$. 

![Diagram of SCC algorithm](image.png)
Proof of lemma 4

- If $d(C) > d(C')$, when first vertex $y$ discovered in $C'$, all other vertices in $C'$ are white, and as before $f[y] = f(C')$.
- Vertices of $C$ are also white, and because of edge $(u, v)$ from $C$ to $C'$, no vertices of $C$ are reachable from $y$, so their discovery times and finishing times are $> f[y]$.
- Thus $f(C) > f(C')$. 

\[\begin{array}{c}
\text{C} \\
\text{C}'
\end{array} \]
Corollary 5

For two components $C, C'$, if there is an edge from $C'$ to $C$ in $G^T$, then $f(C) > f(C')$.

Thus the component first visited in the DFS search on $G^T$ has no edge to any other component.
Conclusion of proof

Can now use induction on the number of trees visited in second DFS to show each one is a separate component