**Directed Graphs**

- Detecting cycles:
  - Linear time $O(|V|+|E|)$
  - Pre-pair numbers

**Topological Sorting**

1. Acyclic directed graph $O(|V|+|E|)$
2. Order the nodes of the DAG in dependence order

**Scheduling**
\[ B \rightarrow D \rightarrow A \rightarrow C = e \quad \text{sorted order} \]

Alg 1 (6)

1. DFS pre-pour numbering \( O(1V_1 + 1E_1) \)
2. If cycle detected, return not sortable
3. Output the nodes in decreasing order of pour numbers \( O(1V_1, 1V_1) \)

- For any node \( u \), its pour number must be greater than all other nodes reachable/descendants.
- \( u \) must come before all such nodes in the topological sort.

b) If \( u \) has an incoming edge, it must be a cross edge from \( x \) since there are no cycles.

\[ \text{pour}_x > \text{pre}_x \Rightarrow \text{pour}_u \]

\( x \) has a higher pour number than \( u \)

\( O(1V_1 + 1E_1) \) time

\( \Rightarrow \) What about sorting by pour numbers \( O(1V_1 \log 1V_1) \) time?

- Add the node to a list as we determine the pour number

Post Order List

\( A [0, 3] \quad B [8, 11] \quad C [1, 6] \quad D [7, 10] \quad E \quad F \quad C \quad A \)
**Strongly Connected Components (SCCs)**

- Directed graph: 1) a path must exist between every pair of nodes in a SCC
- 2) it must be a maximal set of nodes
  - All mutually reachable nodes must be in SCC

**Approach to list the SCCs**

1. Find a sink SCC in G
2. Delete that SCC
3. Repeat.
Finding sink components is not obvious.

Any source in the reverse graph $G^R$ is a sink in $G$.

1. $O(|V|+|E|)$: given $G$, compute reverse graph $G^R$ do DFS numbering on $G^R$
2. $O(|V|+|E|)$: do connected components from the next unvisited vertex in reverse order $G^R$ post-numbering

Overall $O(|V|+|E|)$ linear.

$G^R$ (reverse graph)

$G$
Identify sink in $G$; not clear how to do.

Apply regular connected components since we cannot escape a sink.

**Breadth first search (BFS)**
- weighted graphs
- shorter paths

Task: find the shortest path from $E$ to all other nodes.

DFS is not good for shorter paths; go deep first.

$O(1+v+e)$

**BFS** $(G, s)$ queue (FIFO)

$d(s) = 0 < distance$

$Q = [s]$

while $Q$ not empty

$u = eject Q$

**DFS** $(G, s)$: stack (FIFO)

$C$, $C$, $C$,
DFS

\[ u = \text{enqueue } Q \]
\[
\text{for all } v \in \text{N}(u) : \\
\text{if not visited}(v) : \\
\quad \text{add } v \text{ to } Q ; \text{ set visited}(v) \\
\quad d(v) = d(v) + 1
\]

Finding shortest paths in weighted graphs

\[ G = (V, E) \]
\[ \omega((u, v)) = \text{weight on edge} \]

Weight function

\[ A^3 - B^3 = 6 \]
\[ A^1 - C^1 = 5 \]

Keep the DFS frontier with shortest distance.