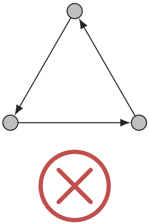
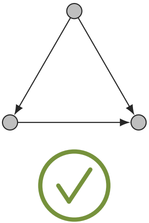
Finding Strongly Connected Components

Directed Acyclic Graphs

Directed Acyclic Graphs (DAG)

A *directed acyclic graph* (or DAG for short) is a directed graph that contains no cycles.





Directed Acyclic Graphs

Cycles + DFS

- A graph contains a cycle if and only if a DFS produces a back edge.
- Thus, if a graph is acyclic, a DFS on this graph produces no back edges.

Lemma

Determining if a given directed graph is a DAG can be done in linear time.

Sources and Sinks

Source and Sink

In a DAG, a *source* is a vertex without incoming edges; a *sink* is a vertex without outgoing edges.

Sources and Sinks

Lemma

Each DAG contains at least one source and one sink.

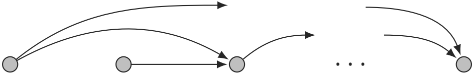
Proof

- Pick arbitrary vertex v.
- If there is a vertex u with $(v, u) \in E$, go to u.
- Repeat this for u.
- Since V is finite and there are no cycles, u will be a sink eventually.
- By symmetry, the same for sources. $\square$

Topological Order

Topological Order

For a directed graph, a vertex order $\langle v_1, v_2, \dots, v_n \rangle$ ($v_i \neq v_j \leftrightarrow i \neq j$) is a *topological order* if $(v_i, v_j) \in E$ implies $i < j$.



Topological Order

Lemma

A graph is a DAG if and only if admits a topological order.

Proof ($\Leftarrow$)

- If a graph admits a topological order, it cannot contain cycles. $\square$

Proof ($\Rightarrow$)

- After removing or adding a source or a sink from or to a DAG, the resulting graph is still a DAG.
- The first vertex of a topological order is a source, the last is a sink.
- Thus, by induction, each DAG admits a topological order. $\square$

Finding a Topological Order

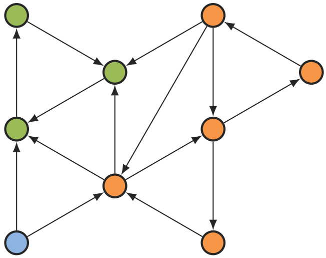
- A post-order on a DFS-tree gives a reversed topological order.

Strongly Connected Components

Strongly Connected Component

Strongly Connected Component

A directed graph is *strongly connected* if every vertex is reachable from every other vertex. A *strongly connected component* is a maximal subgraph which is strongly connected.

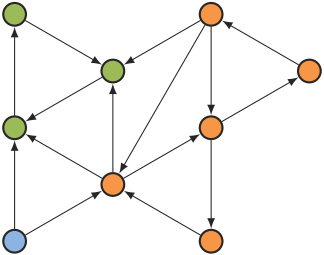


A graph with three strongly connected components.

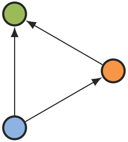
SCC-Graph $\mathcal{S}(G)$

SCC-Graph $\mathcal{S}(G)$

- Assume, G has SCCs $S_1, S_2, \ldots, S_k$.
- For each SCC S_i in G, create a vertex v_i in $\mathcal{S}(G)$.
- Add an edge (v_i, v_j) to $\mathcal{S}(G)$, if there are two vertices u_i and u_j in G with $u_i \in S_i$, $u_j \in S_j$ and $(u_i, u_j) \in E$.



G



$\mathcal{S}(G)$

SCC-Graph $\mathcal{S}(G)$

Lemma

For a directed graph G, $\mathcal{S}(G)$ is acyclic.

Lemma

All vertices in an SCC S are descendants of its first vertex in a DFS-tree.

If v is the first vertex of a SCC S in a DFS-tree, we will call v the *root* of S.

Conclusion

- SCCs have topological order
- Post-order of roots in DFS-tree (of G) gives topological order of SCCs

Sink-SCC

Assume, SCC S is a sink in $\mathcal{S}(G)$ and has root v. Let $\mathcal{D}[v]$ be the descendants of v (including v).

Observations

- There are no edges from S to another SCC.
- For each $u \in \mathcal{D}[v]$ ($u \neq v$), there is a path back to v.

Theorem

A vertex v is the root of a sink S in $\mathcal{S}(G)$ if and only if, for all $u \in \mathcal{D}[v]$,

(i) $(u, w) \in E$ implies $w \in \mathcal{D}[v]$, and

(ii) if $u \neq v$, there is an $x \in \mathcal{D}[u]$ with $(x, y) \in E$ and $y \notin \mathcal{D}[u]$.

Proof of Theorem

$(\longrightarrow)$

- S is sink, i. e., for all $(u, w) \in E$, $u \in S$ implies $w \in S$ and $u \in \mathcal{D}[v]$.
- If $u \neq v$, then there is a path back to v. Thus, u has descendant x with $(x, y) \in E$, $y \notin \mathcal{D}[u]$.

$(\longleftarrow)$

(1) Assume, v is not root.
- There is a path P from v to an ancestor r.
- Thus, there is an edge (u, w) where u is descendant and w is not.

(2) Assume S is not sink.
- There is another SCC reachable from v which is sink and has a root r.
- Because of (i) and $(\longrightarrow)$, $r \in \mathcal{D}[v]$ and, for all $x \in \mathcal{D}[r]$, $(x, y) \in E$ implies $y \in \mathcal{D}[r]$. (Contradiction with (ii))

Algorithm – Identify a Root of a Sink

Assume there is a $u \in \mathcal{D}[v]$ with $(u, w) \in E$ and $w \notin \mathcal{D}[v]$

- (u, w) is cross or back edge.
- Therefore, w was visited in DFS before v.

Lowpoint $\operatorname{low}(v)$

- The lowest pre-order index $\operatorname{pre}(w)$ of a vertex w which is in the (outgoing) neighbourhood of any descendant of v (including v).
- $\operatorname{low}(v) := \min\big(\operatorname{pre}(v), \min\{\operatorname{pre}(w) \mid (u, w) \in E, u \in \mathcal{D}[v]\}\big)$
- Computable with post-order traversal.

Theorem

A vertex v is the root of a sink S in $\mathcal{S}(G)$ if and only if

(i) $\operatorname{pre}(v) \leq \operatorname{low}(v)$ and

(ii) $\operatorname{pre}(u) > \operatorname{low}(u)$ for all $u \in \mathcal{D}[v]$ with $u \neq v$.

Algorithm – Identify Remaining Roots

Naive strategy

- Identify roots of sinks. Descendant of v (including v) are corresponding SCCs.
- Remove corresponding SCCs from graph.
- Repeat.

Observation

- We identify roots by post-order.
- In a DAG, the first vertex in post-order is last vertex in topological order.
- Therefore, we process a root of a sink before all other roots.

New strategy

- After identifying first root, “remove”† corresponding sink before continuing with DFS.

† Flagging as removed and ignore later is sufficient.