Strongly Connected Components

Let $G = (V, E)$ be a directed graph

Write $u \rightsquigarrow v$ if there is a path from $u$ to $v$ in $G$

Write $u \sim v$ if $u \rightsquigarrow v$ and $v \rightsquigarrow u$

$\sim$ is an equivalence relation:

- $u \sim u$
- $u \sim v$ implies $v \sim u$
- $u \sim v$ and $v \sim w$ implies $u \sim w$

$\sim$’s equivalence classes are called the strongly connected components (SCC’s) of $G$

For $v \in V$, $C(v) := v$’s SCC
The component graph

Idea: collapse each SCC’s into a single node

Formally: component graph $G^{\text{scc}} = (V^{\text{scc}}, E^{\text{scc}})$

$V^{\text{scc}} = \text{the SCC’s } C_1, \ldots, C_k \text{ of } G$

$E^{\text{scc}} = \{(C_i, C_j) : i \neq j, (u, v) \in E \text{ for some } u \in C_i, v \in C_j\}$
Lemma 1. $u \leadsto v$ in $G \iff C(u) \leadsto C(v)$ in $G^{\text{SCC}}$
Lemma 2. $G^{scc}$ is acyclic.

- Suppose there is a cycle.
- By definition, no self loops in $G^{scc}$, so the cycle must contain two distinct nodes, say $C(u)$ and $C(v)$.
- Then we have $C(u) \rightarrow C(v)$ and $C(v) \rightarrow C(u)$ in $G^{scc}$.
- By Lemma 1, $u \rightarrow v$ and $v \rightarrow u$ in $G$.
- Thus, $C(u) = C(v) \Rightarrow \Leftarrow$
- QED
An application

Scheduling with constraints:

- We want to schedule a set of tasks
- Each task is represented by a node in a directed graph $G$
- Edges in $G$ represent scheduling constraints: if
  - $\nu$ and $\omega$ are distinct tasks,
  - there is a path from $\nu$ to $\omega$ in $G$, and
  - both $\nu$ and $\omega$ are performed,
    then $\nu$ must be performed before $\omega$
- Each task has a profit associated with it: we want to schedule tasks to maximize profit
A solution:
1. Compute component graph, topologically sorted
2. For each SCC, select the task with maximum profit
3. Output the tasks from Step 2 in topological order

Profits: \( a = 1, b = 2, c = 3, \) etc.

Optimal schedule: \( e, d, g, h \)
Special case: $G$ is undirected
\[(u, v) \in E \iff (v, u) \in E\]

SCC’s are just called *connected components*

The component graph consists of isolated nodes — no edges between components

Easy to compute: the trees in the DFS forest are the connected components
Computing SCC’s: the Kosaraju/Sharir Algorithm

For a graph $G$, let $G^T$ denote its “transpose” or “reverse” — same as $G$ but with all edges reversed.

$G$ and $G^T$ have the same SCC’s — in fact, $(G^T)^{scc} = (G^{scc})^T$

Algorithm $SCC(G)$:

1. call $DFS(G)$, and order the nodes $v_1, \ldots, v_n$ in order of decreasing finishing time (as in $DFSTopSort$)
2. compute $G^T$
3. call $DFS(G^T)$ — but in the top-level loop, process in the order $v_1, \ldots, v_n$
   the trees in the DFS forest are the SCC’s of $G$

Running time: $O(|V| + |E|)$
Example:

\[ G : \]

\[ G_T : \]

\[ G^{\text{SCC}} : \]
Notation: let $f[u]$ be the finish time in the first DFS, and let $f(U) := \max \{f[u] : u \in U\}$

Lemma 3. Suppose $(C, C') \in E^{\text{sc}}$. Then $f(C) > f(C')$

Proof. In the first DFS, let $x$ be the first node discovered in $C \cup C'$

Case 1: $x \in C$

By the White Path Theorem, all nodes in $C \cup C'$ are descendents of $x$ in the DFS forest

By the Parenthesis Theorem, $f[x] = f(C) > f(C')$
Case 2: $x \in C'$

By the White Path Theorem, all nodes in $C'$ are descendents of $x$ in the DFS forest.

By Lemma 2, there is no path from $C'$ to $C$ in $G^{scc}$, and so no node in $C$ is reachable from $x$.

so at time $f[x]$, all nodes in $C$ are still white.

$\therefore f(C) > f[x] = f(C')$. QED
**Theorem.** Algorithm SCC is correct.

*Proof.* Let $T_1, \ldots, T_\ell$ be the trees of the DFS forest created in step 3.

Let $C_1, \ldots, C_k$ be the SCC’s, with $f(C_i) > f(C_{i+1})$.
At step 3, we start with a vertex $x_1$ in $C_1$.
By White Path Theorem, all nodes in $C_1$ will be in $T_1$.
By Lemma 3, in $G^T$, there are no edges leaving $C_1$.
∴ the nodes of $C_1$ are exactly the nodes of $T_1$. 

\begin{center}
\begin{tikzpicture}
\node[shape=circle,draw=black] (A) at (0,0) {$C_1$};
\node[shape=circle,draw=black] (B) at (1,1) {$C_2$};
\node[shape=circle,draw=black] (C) at (2,1) {$C_3$};
\node[shape=circle,draw=black] (D) at (3,0) {$C_4$};
\path[->,thick]
(A) edge (B)
(B) edge (C)
(C) edge (D)
(D) edge (C)
(A) edge (B);
\end{tikzpicture}
\end{center}
Next, we pick a node in $C_2$, and at this time, all nodes in $C_1$ are black, and all nodes in $C_2, \ldots, C_k$ are white.

By White Path Theorem, $T_2$ contains all nodes in $C_2$, and by Lemma 3, $T_2$ contains no other nodes.

$\therefore$ the nodes of $C_2$ are exactly the nodes of $T_2$.

Proceeding by induction, we get $T_i = C_i$ for $i = 1, \ldots, \ell$, and so $k = \ell$. QED.
Representation of $G^{\text{SCC}}$

- Let $C_1, \ldots, C_k$ be the SCC’s
- Number the nodes $1 \ldots k$
- Standard adjacency list representation of $G^{\text{SCC}}$
- Also:
  - An array mapping $v \in V$ to $j \in \{1, \ldots, k\}$, where $v \in C_j$
  - An array mapping $j \in \{1, \ldots, k\}$ to a list representation of $C_j$
- This can all be done in time $O(|V| + |E|)$, and we may assume that $C_1, \ldots, C_k$ are already in topological order — in fact Algorithm SCC outputs $C_1, \ldots, C_k$ in topological order
Another application

**Problem:** A graph $G = (V, E)$ is called *semi-connected* if for all $u, v \in V$, $u \rightarrow v$ or $v \rightarrow u$.

Show how to test if $G$ is semi-connected in time $O(|V| + |E|)$
First consider the problem for DAG’s

Let $v_1, \ldots, v_n$ be a topological sort of $G$

Claim: $G$ is semi-connected $\iff$ there is an edge $v_i \rightarrow v_{i+1}$ for each $i = 1 \ldots n - 1$
Now consider a general graph

Claim: $G$ is semi-connected $\iff G^{\text{scc}}$ is semi-connected (follows directly from Lemma 1)

Algorithm:

1. Run algorithm $\text{SCC}$ to get $G^{\text{scc}}$ (which is a DAG)
2. Test if $G^{\text{scc}}$ is semi-connected (as above)

**General principle:** Try to reduce questions about graphs to questions about DAG’s.
Another application

Consider the “gathering coins” problem for a general directed graph

- Given a directed graph $G = (V, E)$
- On each node $v$ there are $N[v]$ coins
- Goal: find the max number of coins that can be gathered on any one path through $G$
- The path need not be simple, but once you pick up the coins on a node, they are gone
We already know how to solve this for a DAG
For a general graph, we start by computing $G^{scc}$
For each SCC $C$, we assign to it $\sum_{v \in C} N[v]$ coins
Now run the DAG algorithm on $G^{scc}$