1 Strongly Connected Components

In this section, we will be discussing only directed graphs. This is another application of DFS.
Consider this problem:
Input: A directed graph $G = (V, E)$
Output: True/False is G "strongly connected"?

Definition 1.1. *A directed graph is* strongly connected *if for any $x, y \in V$ there is a path from x to y and a path from y to x.*

How can we tell if a graph is strongly connected or not?

The obvious ways to test if a graph is strongly connected or not is to run BFS or DFS from each vertex, and see if there is a path connecting all pairs of vertices.

Another way would be to compute the transitive closure of the graph.
These methods require time $O(|V|^2 + |V||E|)$ and $O(|V|^3 \log(|V|))$, and it might seem like this can't be beat because we are testing whether or not all pairs of vertices have this property.

However, we can solve this problem in time $O(|V| + |E|)$ with the following algorithm:

Strongly-Connected-Algorithm
Step 1: Pick a vertex $v \in V$ and use DFS to check if every vertex can be reached from v. If not, return false.
Step 2: Compute G^T, the transpose graph of G.
Step 3: Do DFS in G^T to check if every vertex can be reached from v. If not, return false. Otherwise, return true.

Each step of this algorithm takes at most $O(|V| + |E|)$ time, and so the running time of the entire algorithm is $O(|V| + |E|)$.

I will outline here a proof of the correctness of the algorithm:

Proof. If the algorithm outputs "false" in step 1, it must have done so because there was some vertex unreachable from v, and so clearly the graph is not strongly connected.

If the algorithm outputs "false" in step 3, there must be some vertex x unreachable from v in G^T, which then means that in G there is no path from x to v.

If the algorithm outputs "true", then for any pair of vertices x and y, the DFS algorithms must have found paths from v to x, from v to y, from x to v, and from y to v.

These paths can be combined to find a path from x to y and a path from y to x. □

Most directed graphs will not be strongly connected, and so we often want to consider the following notion of connectedness.

Definition 1.2. *Let $G = (V, E)$ be a directed graph. A* strongly connected component (SCC) *of G is a maximal subset of vertices that induces a strongly connected subgraph.*

Some terms here probably need explaining.

By saying the subset is maximal, we mean that no additional vertices can be added without breaking the "induces a strongly connected subgraph" property.

For a set of vertices, the induced subgraph is the graph made by taking the vertices of the set and all edges of G connecting those vertices.
Put simply, the strongly connected components of a graph are the "islands" of connected vertices in the graphs.

Consider the following algorithm for finding the SSCs of a graph:
SCC-Algorithm-1
Step 1: Pick a vertex $v \in V$ and use DFS to compute the following set:

$$A_v = \{u \in V : \text{There is a path from } v \text{ to } u \text{ in } G\}$$

Step 2: Use DFS in G^T to compute the following set:

$$B_v = \{u \in V : \text{There is a path from } u \text{ to } v \text{ in } G\}$$

Step 3: $A_v \cap B_v$ is a SCC of G. Delete these vertices from G, and goto step 1 until all vertices have been deleted from G.

The since the graph could have $|V|$ SCCs, the running time of this algorithm is $O(|V|^2 + |V| \cdot |E|))$, which is quite slow. To improve on this, we need to deepen out understanding of SCCs.
Given some graph G, suppose we construct a graph G^{SCC} in this way:

G^{SCC} has a vertex representing each SCC of G,

and any two vertices x, y of G^{SCC} are connected by an edge from x to y if there is an edge connecting a vertex of the SCC x represents to a vertex of the SCC y represents.

G^{SCC} has a property useful to us; it is acyclic, and therefore has a topological sort.
This will be crucial in proving that the following algorithm works correctly:

SCC-Algorithm-2
Step 1: Do DFS on G, record the finishing times.
Step 2: Compute G^T
Step 3: Do DFS on G^T, processing vertices in descending order of finishing times from step 1 (i.e. modify DFS so that whereever it arbitrarily selected vertices it now selects vertices based on highest finishing times.)
Step 4: Identify the "trees" from step 3, i.e. delete all back, forward, and cross edges identified by DFS in step 3. The trees are the connected components that remain, and each tree is a SCC of G.

To prove the correctness of this algorithm, we will need the following lemmas and definition:

Lemma 1.3. *G^{SCC} has a topological sort.*

Proof. Assume there is a cycle in G^{SCC}.
Let $a_1, a_2, \ldots, a_k$ be the vertices in this cycle.
Let $A_1, A_2, \ldots, A_k \subseteq V$ be the SCCs associated with $a_1, a_2, \ldots, a_k$.
Then for all $x, y \in A_1 \cup A_2 \cup \ldots \cup A_k$ there is a path from x to y and y to x (just need to travel along the cycle).
So A_1 is not maximal.
$\rightarrow\leftarrow$
So G^{SCC} is acyclic. □

Lemma 1.4. *G and G^T have the same SCCs.*

Proof. For all vertices x and y, reversing the direction of the edges does not change if there are or are not a paths from x to y and from y to x. □

Definition 1.5. *For $W \subseteq V$, let*

$$\begin{aligned} \mathit{start}(W) &= \min_{w \in W}\{w.d\} \\ \mathit{finish}(W) &= \max_{w \in W}\{w.f\} \end{aligned}$$

where discovery and finish are those parameters found by a particular instance of DFS.

Lemma 1.6. *Let C, C' be distinct SCCs of G, and suppose $\exists (u, v) \in E$ where $u \in C$ and $v \in C'$. Then finish(C) > finish(C').*

Proof. First consider the case where $\text{start}(C) > \text{start}(C')$, then the case where Suppose $\text{start}(C) < \text{start}(C')$. □

Combining these lemmas, we can see why the algorithm works.

When we do DFS in step 1, the SCC at the beginning of the topological sort of G^{SCC} will have the highest finishing time.

So when we begin our DFS in step 3, we will start in the first SCC of the topological sort and will be unable to escape this SCC, since the edges are reversed in G^T.

Since the SCCs of G are the same as the SCCs of G^T, the DFS finds exactly the vertices of the SCC.

When the DFS picks the next starting point, the exploration will again be restricted to exactly one SCC, because of the change in direction of edges and the SCCs that have already been discovered.