Strongly connected components

- Definition and motivation
- Algorithm

Chapter 22.5 in the textbook (pp 552—557).

Connected components

- Find the largest components (sub-graphs) such that there is a path from any vertex in it to any other vertex.
- Applications: networking, communications.
- Undirected graphs: apply BFS/DFS (inner function) from a vertex, and mark vertices as visited. Upon termination, repeat for every unvisited vertex.
- Directed graphs: strongly connected components, not just connected: a path from u to v AND from v to u, which are not necessarily the same!

Example: strongly connected components

Example: strongly connected components

Strongly connected components graph

- Definition: the strongly connected components (SCC) $C_1, \ldots, C_k$ of a directed graph $G = (V,E)$ are the largest disjoint sub-graphs (no common vertices or edges) such that for any two vertices $u$ and $v$ in $C_i$, there is a path from $u$ to $v$ and from $v$ to $u$.
- Equivalence classes of the binary path$(u,v)$ relation, denoted by $u \sim v$. The relation is not symmetric!
- Goal: compute the strongly connected components of $G$ in linear time $\Theta(|V|+|E|)$.

Strongly connected components graph

- Definition: the SCC graph $G^* = (V^*,E^*)$ of the graph $G = (V,E)$ is as follows:
  - $V^* = \{C_1, \ldots, C_k\}$. Each SCC is a vertex.
  - $E^* = \{(C_i,C_j)\mid ij \text{ and } (x,y) \in E, \text{ where } x \in C_i \text{ and } y \in C_j\}$. A directed edge between components corresponds to a directed edge between them from any of their vertices.
  - $G^*$ is a directed acyclic graph (no directed cycles)!
- Definition: the transpose graph $G^T = (V,E^T)$ of the graph $G = (V,E)$ is $G$ with its edge directions reversed: $E^T = \{(u,v) \mid (v,u) \in E\}$.
Example: SCC graph

Example: transpose graph $G^T$

Data Structures, Spring 2004 © L. Joskowicz

SCC algorithm

Idea: compute the SCC graph $G^* = (V^*, E^*)$ with two DFS, one for $G$ and one for its transpose $G^T$, visiting the vertices in reverse order.

SCC($G$)

1. DFS($G$) to compute finishing times $f[v], \forall v \in V$
2. Compute $G^T$
3. DFS($G^T$) in the order of decreasing $f[v]$
4. Output the vertices of each tree in the DFS forest as a separate SCC.

Example: computing SCC (1)

Example: computing SCC (2)

Example: computing SCC (3)
Example: computing SCC (4)

Example: computing SCC (5)

Example: computing SCC (6)

Example: computing SCC (2)

Labeled transpose graph $G^T$

Proof of correctness: SCC (1)

Lemma 1: Let $C$ and $C'$ be two distinct SCC of $G = (V,E)$, let $u,v \in C$ and $u',v' \in C'$. If there is a path from $u$ to $u'$, then there cannot be a path from $v$ to $v'$.

Definition: the start and finishing times of a set of vertices $U \subseteq V$ is:

$d[U] = \min_{u \in U} \{d[U]\}$

$f[U] = \min_{u \in U} \{f[U]\}$

Proof of correctness: SCC (2)

Lemma 2: Let $C$ and $C'$ be two distinct SCC of $G$, and let $(u,v) \in E$ where and $u \in C$ and $v \in C'$. Then, $f[C] > f[C']$.

Proof: there are two cases, depending on which strongly connected component, $C$ or $C'$ is discovered first.

1. $C$ was discovered before $C'$: $d[C] < d[C']$

Let $x$ be the first vertex discovered in $C$. There is a path in $G$ from $x$ to each vertex of $C$ which has not yet been discovered. Because $(u,v) \in E$, for any vertex $w \in C'$, there is also a path at time $d[x]$ from $x$ to $w$ in $G$ consisting only of unvisited vertices: $x \rightarrow u \rightarrow v \rightarrow w$. Thus, all vertices in $C$ and $C'$ become descendants of $x$ in the depth-first tree. Therefore, $f[x] = f[C] > f[C']$. 
Proof of correctness: SCC (3)

2. \( \delta(C) < \delta(C') \)

Let \( y \) be the first vertex discovered in \( C' \). At time \( \delta(y) \), all vertices in \( C' \) are unvisited. There is a path in \( G \) from \( y \) to each vertex of \( C' \) which has only vertices not yet discovered. Thus, all vertices in \( C' \) will become descendants of \( y \) in the depth-first tree, and so \( f[y] = f[C] \). At time \( \delta(y) \), all vertices in \( C \) are unvisited. Since there is an edge \((u,v)\) from \( C \) to \( C' \), there cannot, by Lemma 1, be a path from \( C' \) to \( C \). Hence, no vertex in \( C \) is reachable from \( y \). At time \( f[y] \), therefore, all vertices in \( C \) are unvisited. Thus, no vertex in \( C \) is reachable from \( y \). At time \( f[y] \), therefore, all vertices in \( C \) are still unvisited.

Proof of correctness: SCC (4)

**Corollary:** for edge \((u,v)\) \( \in E \), and \( u \in C \) and \( v \in C' \)

\[ f[C] < f[C'] \]

This provides the clue to what happens during the second DFS.

The algorithm starts at \( x \) with the SCC \( C \) whose finishing time \( f[C] \) is maximum. Since there are no vertices in \( G^T \) from \( C \) to any other SCC, the search from \( x \) will not visit any other component!

Once all the vertices have been visited, a new SCC is constructed as above.

Proof of correctness: SCC (3)

When \( u \) is visited, all the vertices \( v \) in its SCC have not been visited. Therefore, all vertices \( v \) are descendants of \( u \) in the depth-first tree.

By the inductive hypothesis, and the corollary, any edges in \( G^T \) that leave \( C \) must be in SCC that have already been visited. Thus, no vertex in any SCC other than \( C \) will be a descendant of \( u \) during the depth first search of \( G^T \). Thus, the vertices of the depth-first search tree in \( G^T \) that is rooted at \( u \) form exactly one connected component.

Proof of correctness: SCC (4)

**Theorem:** The SCC algorithm computes the strongly connected components of a directed graph \( G \).

**Proof:** by induction on the number of depth-first trees found in the DFS of \( G^T \); the vertices of each tree form a SCC. The first \( k \) trees produced by the algorithm are SCC.

**Basis:** for \( k = 0 \), this is trivially true.

**Inductive step:** The first \( k \) trees produced by the algorithm are SCC. Consider the \((k+1)^{th} \) tree rooted at \( u \) in SCC \( C \). By the lemma, \( f[u] = f[C] > f[C'] \) for SCC \( C' \) that has not yet been visited.

Uses of the SCC graph

- **Articulation:** a vertex whose removal disconnects \( G \).
- **Bridge:** an edge whose removal disconnects \( G \).
- **Euler tour:** a cycle that traverses all edges of \( G \) exactly once (vertices can be visited more than once)

All can be computed in \( O(|E|) \) on the SCC.