**Finding strongly-connected components**

Tyler Moore  
CSE 3353, SMU, Dallas, TX

Lecture 9

---

**Strongly connected components**

A **strongly connected component** is the maximal subset of a graph with a directed path between any two vertices.

---

**Strong connectivity**

**Def.** Nodes $u$ and $v$ are **mutually reachable** if there is a both path from $u$ to $v$ and also a path from $v$ to $u$.

**Def.** A graph is **strongly connected** if every pair of nodes is mutually reachable.

**Lemma.** Let $s$ be any node. $G$ is strongly connected iff every node is reachable from $s$, and $s$ is reachable from every node.

**Pf.**  

$\Rightarrow$ Follows from definition.  

$\Leftarrow$ Path from $u$ to $v$: concatenate $u \rightarrow s$ path with $s \rightarrow v$ path.  

Path from $v$ to $u$: concatenate $v \rightarrow s$ path with $s \rightarrow u$ path.  

---

**Strong connectivity: algorithm**

**Theorem.** Can determine if $G$ is strongly connected in $O(m + n)$ time.  

**Pf.**

- Pick any node $s$.  
- Run BFS from $s$ in $G$.  
- Run BFS from $s$ in $G^\text{reverse}$.  
- Return true iff all nodes reached in both BFS executions.  
- Correctness follows immediately from previous lemma.
**Strong components**

**Def.** A strong component is a maximal subset of mutually reachable nodes.

**Theorem.** [Tarjan 1972] Can find all strong components in $O(m + n)$ time.

---

**Strongly connected components**

A **strongly connected component** is the maximal subset of a graph with a directed path between any two vertices.

---

**Is Wikipedia a strongly connected graph?**

---

**Strongly connected components**

---

**Are supernodes in a DAG?**
Strongly connected components

What if we transpose all edges?

SCCs don't change

Kosaraju’s algorithm for finding SCCs

1. Get a topological sort of all vertices
2. Transpose the graph (reverse all edges)
3. Traverse the graph in topologically sorted order, adding an SCC each time a dead end is reached.

Kosaraju’s Algorithm for Finding Strongly Connected Components

1. Get a topological sort of all vertices
   - topsort: [a, b, e, f, g, c, d, h, i]
   - seen: {}
   - sccs: []
2. Reverse all edges
   - topsort: [a, b, e, f, g, c, d, h, i]
   - seen: {}
   - sccs: []
Kosaraju’s Algorithm for Finding Strongly Connected Components

3. Traverse the graph in topologically sorted order, adding an SCC each time a dead end is reached.

- topsort: [a, b, e, f, g, c, d, h, i]
- seen: {a,b,c,d}
- sccs: [{a,b,c,d}]

1st SCC

- topsort: [a, b, e, f, g, c, d, h, i]
- seen: {a,b,c,d,e,g,f}
- sccs: [{a,b,c,d}, {e,g,f}]

2nd SCC

- topsort: [a, b, e, f, g, c, d, h, i]
- seen: {a,b,c,d,e,g,f}
- sccs: [{a,b,c,d}, {e,g,f}]

3rd SCC

---

Code for Kosaraju’s SCC Algorithm

```python
def tr(G):
    GT = {}
    for u in G:
        GT[u] = set()
    for u in G:
        for v in G[u]:
            GT[v].add(u)
    return GT

def scc(G):
    GT = tr(G)
    sccs, seen = [], set()
    for u in iter_dfs_topsort(G):  # DFS starting points
        if u in seen: continue      # Ignore covered nodes
        C = walk(GT, u, seen)       # Don’t go “backward” (seen)
        seen.update(C)             # We’ve now seen C
        sccs.append(C)             # Another SCC found
    print(sccs)
```

---
Exercise 1: Apply Kosaraju’s SCC Algorithm

Graph $G$

- What is the topological sort of $G$? Let’s make the DFS tree starting from $a$
- What are the strongly connected components?

Graph $G$

$\begin{array}{c}
   a & \rightarrow & c & \rightarrow & e \\
   \downarrow & & \downarrow & & \downarrow \\
   b & \rightarrow & d & \leftarrow & f \\
\end{array}$

Exercise 2: Apply Kosaraju’s SCC Algorithm

Graph $G$

- What is the topological sort of $G$? Let’s make the DFS tree starting from $a$
- What are the strongly connected components?

Graph $G$

$\begin{array}{c}
   a & \rightarrow & d & \rightarrow & g \\
   \uparrow & & \uparrow & & \uparrow \\
   b & \leftarrow & e & \rightarrow & h \\
   \downarrow & & \downarrow & & \downarrow \\
   c & \leftarrow & f & \rightarrow & i \\
\end{array}$

Transpose($G$)

$\begin{array}{c}
   d & \leftarrow & a & \leftarrow & e \\
   \downarrow & & \downarrow & & \downarrow \\
   h & \rightarrow & b & \rightarrow & f \\
   \downarrow & & \downarrow & & \downarrow \\
   i & \rightarrow & c & \rightarrow & g \\
\end{array}$