Further Insights into SCCs

Ru Wang

Department of Computer Science and Engineering
Chinese University of Hong Kong
Given a directed graph $G = (V, E)$, the goal of the strongly connected components problem is to divide $V$ into disjoint subsets, each being an SCC.

Example:

We should output: $\{a, b, c\}$, $\{d, e, f, g, k, l\}$, $\{h, i\}$, and $\{j\}$.
Algorithm

Step 1: Run DFS on $G$ and list the vertices by the order they turn black.

- If a vertex is the $i$-th vertex turning black, define its label as $i$.

Step 2: Obtain the reverse graph $G^{rev}$ by flipping all the edge directions in $G$.

Step 3: Perform DFS on $G^{rev}$ subject to the following rules:

- Rule 1: Start at the vertex with the largest label.
- Rule 2: When a restart is needed, do so from the white vertex with the largest label.

Output the vertices in each DFS-tree as an SCC.

Further Insights into SCCs
Next, we will show how to implement the SCC algorithm in $O(|V| + |E|)$ time. You can assume that $V = \{1, 2, \ldots, n\}$.

**Example:**

![Graph G with nodes 1 to 12 and edges connecting them]
Step 1

Perform DFS on $G$ and record the turn-black order in an array $A$.

- $A[i]$ stores the vertex with label $i$.

Time: $O(|V| + |E|)$. 

Further Insights into SCCs
Step 2

Obtain $G^{\text{rev}} = (V, E^{\text{rev}})$ from $G$ in $O(|V| + |E|)$ time.

We will illustrate how to do so through an example.
Step 2

Initialize the head-pointer array for $G^{rev}$.

adj. list of $G$  
adj. list of $G^{rev}$
Step 2

Scan the neighbor list of each $u \in V$ in $G$. For every out-neighbor $v$ of $u$, add $u$ to the neighbor list of $v$ in $G^{\text{rev}}$.
Step 2

Scan the neighbor list of each $u \in V$ in $G$. For every out-neighbor $v$ of $u$, add $u$ to the neighbor list of $v$ in $G^{rev}$. 

adj. list of $G$  

adj. list of $G^{rev}$
Step 2

Scan the neighbor list of each $u \in V$ in $G$.
For every out-neighbor $v$ of $u$, add $u$ to the neighbor list of $v$ in $G^{rev}$. 

adj. list of $G$ 

adj. list of $G^{rev}$
Step 2

Scan the neighbor list of each $u \in V$ in $G$.
For every out-neighbor $v$ of $u$, add $u$ to the neighbor list of $v$ in $G^{rev}$. 

adj. list of $G$ 

adj. list of $G^{rev}$
Step 3

Perform DFS on $G^{rev}$ and use $A$ to select the vertex to start/restart from.

vertex with label 12

A

vertex with label 12

the 1st starting vertex
Step 3

Start the 1st DFS on $G^{rev}$ from vertex 10. Output \{10\}.

$G^{rev}$

Vertex 10 is now black.
Step 3


Further Insights into SCCs
Step 3

Start the 2nd DFS on $G^{rev}$ from 9. Output $\{8, 9\}$.

$G^{rev}$

Vertices 8 and 9 are now black.
Step 3


![Graph diagram]
Step 3

Start the 3rd DFS on $G^{rev}$ from 7. Output $\{7, 5, 4, 6, 12, 11\}$.

Vertices 7, 5, 4, 6, 12, and 11 are now black.
Step 3


$G^{rev}$
Step 3

Start the 4th DFS on $G^{rev}$ from 1. Output $\{1, 2, 3\}$. 

DFS-tree

$G^{rev}$
Step 3

Scan $A$ backwards from 1 and find no other white vertices. The algorithm finishes.
Next, we will unveil a mathematical structure of the SCC problem that suggests a generic algorithmic paradigm.
An SCC is a **sink SCC** if it has no outgoing edge in $G^{scc}$.

$S_4$ is the only sink SCC in the above example.
A conceptual SCC strategy

1. while \( G^{scc} \) not empty do
2. \( S \leftarrow \) a sink SCC
3. run DFS from any vertex in \( S \)
4. remove all the vertices in \( S \) from \( G \);
delete vertex \( S \) from \( G^{scc} \)

Example:

DFS from anywhere in \( S_4 \) finds SCC \( \{a, b, c\} \).
A conceptual SCC strategy

1. while \( G^{scc} \) not empty do
2. \( S \leftarrow \) a sink SCC
3. run DFS from any vertex in \( S \)
4. remove all the vertices in \( S \) from \( G \);
delete vertex \( S \) from \( G^{scc} \)

Example:

Delete \( S_4 \) from \( G \) and \( G^{scc} \). New sink vertex: \( S_3 \).
A conceptual SCC strategy

1. while $G^{scc}$ not empty do
2. $S \leftarrow$ a sink SCC
3. run DFS from any vertex in $S$
4. remove all the vertices in $S$ from $G$; delete vertex $S$ from $G^{scc}$

Example:

DFS from anywhere in $S_3$ finds SCC \{d, e, f, g, k, l\}.
A conceptual SCC strategy

1. while $G^{scc}$ not empty do
2. \[ S \leftarrow \text{a sink SCC} \]
3. run DFS from any vertex in $S$
4. remove all the vertices in $S$ from $G$; delete vertex $S$ from $G^{scc}$

Example:

A conceptual SCC strategy

1. while $G^{scc}$ not empty do
2. $S \leftarrow$ a sink SCC
3. run DFS from any vertex in $S$
4. remove all the vertices in $S$ from $G$; delete vertex $S$ from $G^{scc}$

Example:

DFS from anywhere in $S_2$ finds SCC $\{i, h\}$. 
A conceptual SCC strategy

1. while $G^{scc}$ not empty do
2. $S \leftarrow$ a sink SCC
3. run DFS from any vertex in $S$
4. remove all the vertices in $S$ from $G$; delete vertex $S$ from $G^{scc}$

Example:  

Delete $S_2$. New sink vertex $S_1$.  

SCC Graph $G^{scc}$
A conceptual SCC strategy

1. while $G^{scc}$ not empty do
2. \( S \leftarrow \) a sink SCC
3. run DFS from any vertex in $S$
4. remove all the vertices in $S$ from $G$; delete vertex $S$ from $G^{scc}$

Example:

The 4th DFS finds SCC \{\(j\}\).
**Question:** Why does our SCC algorithm work on the reverse graph, as opposed to the original one?

**Answer:** Non-trivial to find the next sink SCC.

**Not easy:** You need to find a vertex in $S_4$ first, then a vertex in $S_3$, then one in $S_2$, and finally in $S_1$. 
It turns out that finding the next sink SCC on the reverse graph is **much** easier.

Sink SCC = $S_1$.
DFS from $j$ finds SCC $\{j\}$
It turns out that finding the next sink SCC on the reverse graph is much easier.

Sink SCC = $S_2$.
DFS from anywhere in $S_2$ finds SCC $\{h, i\}$
It turns out that finding the next sink SCC on the reverse graph is much easier.

Sink SCC = $S_3$.

DFS from anywhere in $S_3$ finds SCC $\{d, e, f, g, k, l\}$. 
It turns out that finding the next sink SCC on the reverse graph is much easier.

Sink SCC = $S_4$. The last DFS finds SCC \{a, b, c\}.

This is exactly how our SCC algorithm finds the SCCs.