COMP251: Topological Sort & Strongly Connected Components

Jérôme Waldispühl & Roman Sarrazin-Gendron
School of Computer Science
McGill University

Based on (Cormen et al., 2002)

Based on slides from D. Plaisted (UNC)
Outline

• Recap: DFS & BFS

• Background material
  - Parenthesis theorem
  - White-Path theorem
  - Edge classification

• Direct Acyclic Graphs (DAGs)
  - Definition
  - Topological Sort

• Strongly Connected Components
Recap: Breadth-first Search

- **Input:** Graph $G = (V, E)$, either directed or undirected, and source vertex $s \in V$.
- **Output:**
  - $d[v] = \text{distance (smallest # of edges, or shortest path) from } s \text{ to } v$, for all $v \in V$. $d[v] = \infty$ if $v$ is not reachable from $s$.
  - $\pi[v] = u$ such that $(u, v)$ is last edge on shortest path $s \rightsquigarrow v$.
    - $u$ is $v$’s predecessor.
  - Builds breadth-first tree with root $s$ that contains all reachable vertices.
Recap: BFS Example

Distance from source $d[v]$

Predecessor $\pi[v]$
Recap: Depth-first Search

- **Input:** $G = (V, E)$, directed or undirected. No source vertex given.
- **Output:**
  - 2 **timestamps** on each vertex. Integers between 1 and $2|V|$.
    - $d[v] = \textit{discovery time}$ ($v$ turns from white to gray)
    - $f[v] = \textit{finishing time}$ ($v$ turns from gray to black)
  - $\pi[v]$: predecessor of $v = u$, such that $v$ was discovered during the scan of $u$’s adjacency list.
- Uses the same coloring scheme for vertices as BFS.
Recap: DFS Example

Vertices on the DFS path are descendants of their predecessors (e.g., x is a descendant of v)

Starting time $d(x)$
Finishing time $f(x)$

Note: The direction of the edges on the DFS path have been reversed to represent the predecessor.
Recap: Parenthesis Theorem

**Theorem 1:**
For all $u$, $v$, exactly one of the following holds:

2. $d[u] < d[v] < f[v] < f[u]$ and $v$ is a descendant of $u$.

- Like parentheses:
  
  OK: ( { } ) [ ]   
  Not OK: ( { } ) 
  
  1 2 3 4 5 6   1 2 3 4

**Corollary**

$v$ is a proper descendant of $u$ if and only if $d[u] < d[v] < f[v] < f[u]$. 
White-path Theorem

Theorem 2

\( v \) is a descendant of \( u \) if and only if at time \( d[u] \), there is a path \( u \stackrel{\sim}{\leadsto} v \) consisting of only white vertices (Except for \( u \), which was just colored gray).

Notation: the arrow \( \sim \) represents a path of any length (i.e., sequence of one or more consecutive edges).
Example (white-path theorem)

v, y, and x are descendants of u.
Edge classification with DFS

The red edges show the edges used by the DFS algorithm (i.e., tree edges)
Classification of Edges

- **Tree edge:** in the depth-first forest. Found by exploring \((u, v)\).
- **Back edge:** \((u, v)\), where \(u\) is a descendant of \(v\) (in the depth-first tree).
- **Forward edge:** \((u, v)\), where \(v\) is a descendant of \(u\), but not a tree edge.
- **Cross edge:** any other edge. Can go between vertices in same depth-first tree or in different depth-first trees.

**Theorem 3**
In DFS of a connected undirected graph, we get only tree and back edges. No forward or cross edges. Proof left as an exercise...
Identification of Edges

- Edge type for edge \((u, v)\) can be identified when it is first explored by DFS.

- Identification is based on the color of \(v\).
  - White – tree edge.
  - Gray – back edge.
  - Black – forward or cross edge.
Directed Acyclic Graph

- DAG – Directed graph with no cycles.
- Good for modeling processes and structures that have a **partial order**:
  - $a > b$ and $b > c \implies a > c$.
  - But may have $a$ and $b$ such that neither $a > b$ nor $b > a$.
- Can always make a **total order** (either $a > b$ or $b > a$ for all $a \neq b$) from a partial order.
Example
DAG of dependencies for putting on goalie equipment.

socks -> hose -> pants -> skates -> leg pads
shorts
T-shirt -> chest pad -> sweater -> mask -> catch glove -> blocker

batting glove
Characterizing a DAG

Lemma 1
A directed graph $G$ is acyclic iff a DFS of $G$ yields no back edges.

Proof:
• $(\Rightarrow)$ Show that back edge $\Rightarrow$ cycle.
  – Suppose there is a back edge $(u, v)$. Then $v$ is ancestor of $u$ in depth-first forest (by definition of a back edge).
  – Therefore, there is a path $v \leadsto u$, so $v \leadsto u \leadsto v$ is a cycle.
Lemma 1
A directed graph $G$ is acyclic iff a DFS of $G$ yields no back edges.

Proof (Contd.):

- $(\Leftarrow)$ Show that a cycle implies a back edge.
  - $c$: cycle in $G$; $v$: first vertex discovered in $c$;
    $(u, v)$: preceding edge in $c$.
  - At time $d[v]$, vertices of $c$ form a white path $v \rightsquigarrow u$.
  - By **white-path theorem**, $u$ is a descendent of $v$ in depth-first forest.
  - Therefore, $(u, v)$ is a back edge.
Topological Sort

Want to “sort” a directed acyclic graph (DAG).

Think of original DAG as a partial order.

You may have several valid total orders.

We want a total order that extends this partial order.
Topological Sort

• Performed on a DAG.
• Linear ordering of the vertices of $G$ such that if $(u, v) \in E$, then $u$ appears somewhere before $v$.

**Topological-Sort ($G$)**

1. call DFS($G$) to compute finishing times $f[v]$ for all $v \in V$
2. as each vertex is finished, insert it onto the front of a linked list
3. return the linked list of vertices

**Time:** $\Theta(V + E)$.
Example 1

Linked List:

- A
- B
- C
- D
- E

1/
Example 1

Linked List:
Example 1

Linked List:
Example 1

Linked List:
Example 1

Linked List:

1/4 -> 2/3

D -> E
Example 1

Linked List:

A → B → C → D → E

A → 5/6 → 6/1 → 1/4

D → 1/4 → 2/3

E
Example 1

Linked List:

A

B

C

D

E

5/

1/4

2/3

6/7

1/4

2/3

6/7

C

D

E
Example 1

Linked List:

A

B

C

D

E

5/8

6/7

1/4

2/3
Example 1

Linked List:

A → B → C → D → E
Example 1

The output may change if the choices of vertices is different, but the result will remain valid.
Example 2

socks
shorts
hose
pants
skates
leg pads
T-shirt
chest pad
sweater
mask
catch glove
blocker

26 socks
24 shorts
23 hose
22 pants
21 skates
20 leg pads
14 t-shirt
13 chest pad
12 sweater
11 mask
6 batting glove
5 catch glove
4 blocker
In the sequence of vertices given by the total order.

Correctness (1)

We want to prove: “Linear ordering of the vertices of $G$ such that if $(u, v) \in E$, then $u$ appears somewhere before $v$.”

$\Rightarrow$ We need to show $\text{if } (u, v) \in E, \text{ then } f[v] < f[u]$. Vertices are inserted at the head of the list as soon as they are finished.

When we explore $(u, v)$, what are the colors of $u$ and $v$?

Assume we just discovered $u$, which is thus gray.

Then, what are the possible colors of $v$?

– Can $v$ be gray?
– Can $v$ be white?
– Can $v$ be black?
Correctness (2)

When we explore \((u, v)\), what are the colors of \(u\) and \(v\)?

– Assume \(u\) is gray (by hypothesis, we just discovered it).

– Is \(v\) gray, too?
  
  \(No\), because then \(v\) would be ancestor of \(u\).
  
  \(\Rightarrow (u, v)\) is a back edge (by definition of a back edge).
  
  \(\Rightarrow\) contradiction of Lemma 1 (DAG has no back edges).

– Is \(v\) white?
  
  • Then becomes descendant of \(u\).
  
  • By parenthesis theorem, \(d[u] < d[v] < f[v] < f[u]\).

– Is \(v\) black?
  
  • Then \(v\) is already finished.
  
  • Since we are exploring \((u, v)\), we have not yet finished \(u\).
  
  • Therefore, \(f[v] < f[u]\).
Strongly Connected Components

- $G$ is strongly connected if every pair $(u, v)$ of vertices in $G$ is reachable from one another.
- A **strongly connected component (SCC)** of $G$ is a maximal set of vertices $C \subseteq V$ such that for all $u, v \in C$, both $u \sim v$ and $v \sim u$ exist.
Component Graph

- $G^{\text{SCC}} = (V^{\text{SCC}}, E^{\text{SCC}})$.
- $V^{\text{SCC}}$ has one vertex for each SCC in $G$.
- $E^{\text{SCC}}$ has an edge if there is an edge between the corresponding SCC’s in $G$.

Example:
\[ G^{\text{SCC}} \text{ is a DAG} \]

**Lemma 2**

Let \( C \) and \( C' \) be distinct SCC’s in \( G \), let \( u, v \in C \) & \( u', v' \in C' \), and suppose there is a path \( u \sim u' \text{ in } G \). Then there cannot also be a path \( v' \sim v \text{ in } G \).

**Proof** (by contradiction):

- Assume there is a path \( v' \sim v \text{ in } G \).
- Then, there are paths \( u \sim u' \sim v' \text{ and } v' \sim v \sim u \text{ in } G \).
- Therefore, \( u \) and \( v' \) are reachable from each other, so they are not in separate SCC’s.
**Transpose of a Directed Graph**

- $G^T = \text{transpose} \text{ of directed } G$.
  - $G^T = (V, E^T), E^T = \{(u, v) : (v, u) \in E\}$.
  - $G^T$ is $G$ with all edges reversed.

- Can create $G^T$ in $\Theta(V + E)$ time if using adjacency lists.

- $G$ and $G^T$ have the *same SCC’s*. ($u$ and $v$ are reachable from each other in $G$ if and only if reachable from each other in $G^T$.)
Algorithm to determine SCCs

**SCC(G)**

1. call DFS(G) to compute finishing times \( f[u] \) for all \( u \)
2. compute \( G^T \)
3. call DFS(\( G^T \)), but in the main loop, consider vertices in order of decreasing \( f[u] \) (as computed in first DFS)
4. output the vertices in each tree of the depth-first forest formed in second DFS as a separate SCC

**Time:** \( \Theta(V + E) \).
Example

$G$

\[ a \rightarrow b \rightarrow c \rightarrow d \]
\[ e \rightarrow f \rightarrow g \rightarrow h \]

With a cycle:
\[ h \rightarrow e \]

After the first DFS. We computed all finishing times in $G$. 

```plaintext
G

13/14 → 11/16 → 1/10 → 8/9
12/15 → 3/4 → 2/7 → 5/6
```

a → b → c → d

e → f → g → h
Then, we compute the transpose $G^T$ of $G$ and sort the vertices with the finishing time calculated in $G$. 
Example

\[ G^T \]

\[
(b \ (a \ (e \ e) \ a) \ b) \quad (c \ (d \ d) \ c) \quad (g \ (f \ f) \ g) \quad (h)
\]
How does it work?

• Idea:
  – By considering vertices in second DFS in decreasing order of finishing times from first DFS, we are visiting vertices of the component graph in topologically sorted order.
  – Because we are running DFS on $G^T$, we will not be visiting any $v$ from a $u$, where $v$ and $u$ are in different components.

Recall: the component graph is a DAG!

• Notation:
  – $d[u]$ and $f[u]$ always refer to first DFS.
  – Extend notation for $d$ and $f$ to sets of vertices $U \subseteq V$:
    – $d(U) = \min_{u \in U} \{d[u]\}$ (earliest discovery time)
    – $f(U) = \max_{u \in U} \{f[u]\}$ (latest finishing time)
SCCs and DFS finishing times

Lemma 3
Let C and C’ be distinct SCC’s in G = (V, E). Suppose there is an edge (u, v) ∈ E such that u ∈ C and v ∈ C’. Then, f (C) > f (C’).

Proof:
• Case 1: d(C) < d(C’)
  – Let x be the first vertex discovered in C.
  – At time d[x], all vertices in C and C’ are white. Thus, there exist paths of white vertices from x to all vertices in C and C’.
  – By the white-path theorem, all vertices in C and C’ are descendants of x in depth-first tree.
  – By the parenthesis theorem, f [x] = f (C) > f(C’).
SCCs and DFS finishing times

Lemma 3
Let $C$ and $C'$ be distinct SCC’s in $G = (V, E)$. Suppose there is an edge $(u, v) \in E$ such that $u \in C$ and $v \in C'$. Then $f(C) > f(C')$.

Proof:
• Case 2: $d(C) > d(C')$
  – Let $y$ be the first vertex discovered in $C'$.
  – At $d[y]$, all vertices in $C'$ are white and there is a white path from $y$ to each vertex in $C'$ $\Rightarrow$ all vertices in $C'$ become descendants of $y$. Again, $f[y] = f(C')$.
  – At $d[y]$, all vertices in $C$ are also white.
  – By lemma 2, since there is an edge $(u, v)$, we cannot have a path from $C'$ to $C$.
  – So, no vertex in $C$ is reachable from $y$.
  – Therefore, at time $f[y]$, all vertices in $C$ are still white.
  – Therefore, for all $w \in C$, $f[w] > f[y]$, which implies that $f(C) > f(C')$. 
Corollary 1
Let $C$ and $C'$ be distinct SCC’s in $G = (V, E)$. Suppose there is an edge $(u, v) \in E^T$, where $u \in C$ and $v \in C'$. Then, $f(C) < f(C')$.

Proof:
• $(u, v) \in E^T \Rightarrow (v, u) \in E$.
• Since SCC’s of $G$ and $G^T$ are the same, $f(C') > f(C)$, by Lemma 3.
Correctness of SCC

1) At beginning, DFS visits only vertices in the first SCC

- When we do the second DFS on $G^T$, we start with the SCC $C$ such that $f(C)$ is maximum.
- This second DFS starts from some $x \in C$, and it visits all vertices in $C$.
- Corollary 1 says that since $f(C) > f(C')$ for all $C \neq C'$, there are no edges from $C$ to $C'$ in $G^T$.
- Therefore, DFS will visit only vertices in $C$.
- Which means that the depth-first tree rooted at $x$ contains exactly the vertices of $C$. 
Correctness of SCC

2) *DFS does not visit more than one new SCC at the time*

- The next root in the second DFS is in SCC $C'$ such that $f(C')$ is maximum over all SCC’s other than C.
  - DFS visits all vertices in $C'$, but *the only edges out of $C'$ go to $C$, which we have already visited.*
  - Therefore, the only tree edges will be to vertices in $C'$.
- Iterate the process.
- Each time we choose a root, it can reach only:
  - vertices in its SCC—get tree edges to these,
  - vertices in SCC’s *already visited* in second DFS—get no tree edges to these.