COMP251: Topological Sort & Strongly Connected Components

Jérôme Waldispühl
School of Computer Science
McGill University

Based on (Cormen et al., 2002)

Based on slides from D. Plaisted (UNC)
We prefer to use an adjacency matrix vs a adjacency list to represent a graph when:

- The graph is sparse    ✗
- The graph is dense     ✓
- The graph is a weighted graph    ✗
- The graph is directed    ✗

Representation of graphs

<table>
<thead>
<tr>
<th>Description</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>The graph is sparse</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>The graph is dense</td>
<td>13</td>
<td>86.7%</td>
</tr>
<tr>
<td>The graph is a weighted graph</td>
<td>2</td>
<td>13.3%</td>
</tr>
<tr>
<td>The graph is directed</td>
<td>0</td>
<td>0%</td>
</tr>
</tbody>
</table>
Let $G$ be a directed graph. We explore $G$ using the BFS algorithm. Which of the following assertions are true?

- The best case running time of BFS is $\Omega(V+E)$  ✓ (if connected)
- All vertices at distance $d$ from the source $s$ are visited before vertices at distance $d+1$  ✓
- All vertices of $G$ are visited even if $G$ has disconnected components  ✗
- The source $s$ can be any vertex of $G$  ✓
Recap: Breadth-first Search

- **Input:** Graph \( G = (V, E) \), either directed or undirected, and *source vertex* \( s \in V \).
- **Output:**
  - \( d[v] = \) distance (smallest # of edges, or shortest path) from \( s \) 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 \( s \leadsto v \).
    - \( u \) is \( v \)'s predecessor.
  - Builds breadth-first tree with root \( s \) that contains all reachable vertices.
Recap: BFS Example

Q: ∅
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

Starting time: \(d(x)\)

Finishing time: \(f(x)\)
**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: ( [ ) ] [ ( ) ]

**Corollary**

$v$ is a proper descendant of $u$ if and only if $d[u] < d[v] < f[v] < f[u]$.
Example (Parenthesis Theorem)

(s (z (y (x x) y) (w w) z) s) (t (v v) (u u) t)
White-path Theorem

Theorem 2

\( v \) is a descendant of \( u \) if and only if at time \( d[u] \), there is a path \( u \sim v \) consisting of only white vertices. (Except for \( u \), which was just colored gray.)
v, y, and x are descendants of u.
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 an undirected graph, we get only tree and back edges. No forward or cross edges.
Example (DFS)
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 \(b > c\) \(\Rightarrow\) \(a > c\).
  - 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
- shorts
- hose
- pants
- skates
- leg pads
- 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.
  – 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.
  – 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.

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:
Example 1

Linked List:
Example 1

Linked List:

A → B → C

D

1/
2/3

E

E
Example 1

Linked List:

1/4 → 2/3
Example 1

Linked List:

A → B → C

D → E
Example 1

Linked List:

A → B
C → D → E

D → E
Example 1

Linked List:

```
6/7  ->  1/4  ->  2/3
  C    D    E
```
Example 1

Linked List:

B → C → D → E
Example 1

Linked List:
Example 1

Linked List:
Example 2

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
Correctness Proof

• Just need to show if \((u, v) \in E\), then \(f[v] < f[u]\).

• When we explore \((u, v)\), what are the colors of \(u\) and \(v\)?
  – \(u\) is gray.
  – Is \(v\) gray, too?
    
    No, because then \(v\) would be ancestor of \(u\).
    
    \(\Rightarrow (u, v)\) is 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’re 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

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

- Suppose there is a path $v' \leadsto v$ in $G$.
- Then there are paths $u \leadsto u' \leadsto v'$ and $v' \leadsto v \leadsto u$ in $G$.
- Therefore, $u$ and $v'$ are reachable from each other, so they are not in separate SCC's.
Transpose of a Directed Graph

• $G^T = \text{transpose of directed } G$.
  – $G^T = (V, E^T), E^T = \{(u, v) : (v, u) \in E\}$.
  – $G^T$ 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

$G$

Diagram with nodes $a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$. Edges illustrate connections between these nodes.
Example

G
Example

G

13/14 → 11/16 → 1/10 → 8/9 → 5/6
12/15 → 3/4 → 2/7 → f → g → h
Example

$G^T$

(b (a (e e) a) b) (c (d d) c) (g (f f) g) (h)
Example

abe → cd → fg → 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.

• **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

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, $f[x] = f(C) > f(C')$.

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 $f(C) > f(C')$. 
SCCs and DFS finishing times

Lemma 4
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 $f(C) > f(C')$. 
SCCs and DFS finishing times

Corollary 1
Let $C$ and $C'$ be distinct SCC's in $G = (V, E)$. Suppose there is an edge $(u, v) \in E^\top$, where $u \in C$ and $v \in C'$. Then $f(C) < f(C')$.

Proof:

• $(u, v) \in E^\top \Rightarrow (v, u) \in E$.
• Since SCC's of $G$ and $G^\top$ are the same, $f(C') > f(C)$, by Lemma.
Correctness of SCC

• When we do the second DFS, on $G^T$, start with SCC $C$ such that $f(C)$ is maximum.
  
  – The 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

• The next root chosen 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’ve already visited.
  – Therefore, the only tree edges will be to vertices in $C'$.
• We can continue the process.
• Each time we choose a root for the second DFS, 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.