DFS in Directed Graphs, Strong Connected Components, and DAGs

Lecture 2
January 20, 2011
Algorithmic Problem

Find all SCCs of a given directed graph.

Previous lecture: saw an $O(n \cdot (n + m))$ time algorithm.
This lecture: $O(n + m)$ time algorithm.
Graph of SCCs

Figure: Graph $G$

Meta-graph of SCCs

Let $S_1, S_2, \ldots S_k$ be the SCCs of $G$. The graph of SCCs is $G^{SCC}$.

- Vertices are $S_1, S_2, \ldots S_k$
- There is an edge $(S_i, S_j)$ if there is some $u \in S_i$ and $v \in S_j$ such that $(u, v)$ is an edge in $G$. 
Proposition

For any graph $G$, the graph of SCCs of $G^{\text{rev}}$ is the same as the reversal of $G^{\text{SCC}}$.

Proof.

Exercise.
Proposition

For any graph $G$, the graph $G^{SCC}$ has no directed cycle.

Proof.

If $G^{SCC}$ has a cycle $S_1, S_2, \ldots, S_k$ then $S_1 \cup S_2 \cup \cdots \cup S_k$ is an SCC in $G$. Formal details: exercise.
Part I

Directed Acyclic Graphs
Directed Acyclic Graphs

Definition

A directed graph $G$ is a **directed acyclic graph (DAG)** if there is no directed cycle in $G$. 

![Diagram of a directed acyclic graph with nodes 1, 2, 3, 4 and directed edges between them](image-url)
Sources and Sinks

**Definition**

- A vertex $u$ is a **source** if it has no in-coming edges.
- A vertex $u$ is a **sink** if it has no out-going edges.
Simple DAG Properties

- Every **DAG** $G$ has at least one source and at least one sink.
- If $G$ is a **DAG** if and only if $G^{\text{rev}}$ is a **DAG**.
- $G$ is a **DAG** if and only each node is in its own strong connected component.

Formal proofs: exercise.
Simple DAG Properties

- Every DAG $G$ has at least one source and at least one sink.
- If $G$ is a DAG if and only if $G^{\text{rev}}$ is a DAG.
- $G$ is a DAG if and only each node is in its own strong connected component.

Formal proofs: exercise.
Simple DAG Properties

- Every DAG $G$ has at least one source and at least one sink.
- If $G$ is a DAG if and only if $G^{\text{rev}}$ is a DAG.
- $G$ is a DAG if and only each node is in its own strong connected component.

Formal proofs: exercise.
Simple DAG Properties

- Every DAG $G$ has at least one source and at least one sink.
- If $G$ is a DAG if and only if $G^{\text{rev}}$ is a DAG.
- $G$ is a DAG if and only each node is in its own strong connected component.

Formal proofs: exercise.
Simple DAG Properties

- Every DAG $G$ has at least one source and at least one sink.
- If $G$ is a DAG if and only if $G^{rev}$ is a DAG.
- $G$ is a DAG if and only each node is in its own strong connected component.

Formal proofs: exercise.
A topological ordering/topological sorting of $G = (V, E)$ is an ordering $<$ on $V$ such that if $(u, v) \in E$ then $u < v$. 

Figure: Graph $G$

Figure: Topological Ordering of $G$
Lemma

A directed graph $G$ can be topologically ordered iff it is a DAG.

Proof.

$\Rightarrow$: Suppose $G$ is not a DAG and has a topological ordering $\prec$. $G$ has a cycle $C = u_1, u_2, \ldots, u_k, u_1$. Then $u_1 \prec u_2 \prec \ldots \prec u_k \prec u_1!$ That is... $u_1 \prec u_1$. A contradiction (to $\prec$ being an order).
Not possible to topologically order the vertices.
A directed graph $G$ can be topologically ordered iff it is a **DAG**.

⇐: Consider the following algorithm:

- Pick a source $u$, output it.
- Remove $u$ and all edges out of $u$.
- Repeat until graph is empty.
- Exercise: prove this gives an ordering.

Exercise: show above algorithm can be implemented in $O(m + n)$ time.
Topological Sort: An Example

Output: 1 2 3 4
Topological Sort: An Example

Output: 1 2 3 4
Topological Sort: An Example

Output: 1 2 3 4
Topological Sort: An Example

Output: 1 2 3 4
Topological Sort: An Example

Output: 1 2 3 4
Topological Sort: Another Example
Topological Sort: Another Example
DAGs and Topological Sort

Note: A DAG $G$ may have many different topological sorts.

Question: What is a DAG with the most number of distinct topological sorts for a given number $n$ of vertices?

$\# n$ singletons

Question: What is a DAG with the least number of distinct topological sorts for a given number $n$ of vertices?

MA Linked List
Using DFS...

... to check for Acyclicity and compute Topological Ordering

Question

Given $G$, is it a DAG? If it is, generate a topological sort.

DFS based algorithm:

- Compute $\text{DFS}(G)$
- If there is a back edge then $G$ is not a DAG.
- Otherwise output nodes in decreasing post-visit order.

Correctness relies on the following:

Proposition

$G$ is a DAG iff there is no back-edge in $\text{DFS}(G)$.

Proposition

If $G$ is a DAG and $\text{post}(v) > \text{post}(u)$, then $(u, v)$ is not in $G$. 
Using DFS...
... to check for Acyclicity and compute Topological Ordering

**Question**
Given $G$, is it a **DAG**? If it is, generate a topological sort.

**DFS** based algorithm:
- Compute $\text{DFS}(G)$
- If there is a back edge then $G$ is not a **DAG**.
- Otherwise output nodes in decreasing post-visit order.

Correctness relies on the following:

**Proposition**
$G$ is a **DAG** iff there is no back-edge in $\text{DFS}(G)$.

**Proposition**
If $G$ is a **DAG** and $\text{post}(v) > \text{post}(u)$, then $(u, v)$ is not in $G$. 
Using DFS...

... to check for Acyclicity and compute Topological Ordering

**Question**

Given $G$, is it a **DAG**? If it is, generate a topological sort.

**DFS** based algorithm:

- Compute $\text{DFS}(G)$
- If there is a back edge then $G$ is not a **DAG**.
- Otherwise output nodes in decreasing post-visit order.

Correctness relies on the following:

**Proposition**

$G$ is a **DAG** iff there is no back-edge in $\text{DFS}(G)$.

**Proposition**

If $G$ is a **DAG** and $\text{post}(v) > \text{post}(u)$, then $(u, v)$ is not in $G$. 
Example
Proposition

\( G \) has a cycle iff there is a back-edge in \( \text{DFS}(G) \).

Proof.

If: \((u, v)\) is a back edge implies there is a cycle \( C \) consisting of the path from \( v \) to \( u \) in \( \text{DFS} \) search tree and the edge \((u, v)\).

Only if: Suppose there is a cycle \( C = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k \rightarrow v_1 \). Let \( v_i \) be first node in \( C \) visited in \( \text{DFS} \). All other nodes in \( C \) are descendents of \( v_i \) since they are reachable from \( v_i \). Therefore, \((v_{i-1}, v_i)\) (or \((v_k, v_1)\) if \( i = 1 \)) is a back edge.
Proposition

\( G \) has a cycle iff there is a back-edge in \( \text{DFS}(G) \).

Proof.

If: \((u, v)\) is a back edge implies there is a cycle \( C \) consisting of the path from \( v \) to \( u \) in \( \text{DFS} \) search tree and the edge \( (u, v) \).

Only if: Suppose there is a cycle \( C = v_1 \to v_2 \to \ldots \to v_k \to v_1 \). Let \( v_i \) be first node in \( C \) visited in \( \text{DFS} \). All other nodes in \( C \) are descendents of \( v_i \) since they are reachable from \( v_i \).

Therefore, \((v_{i-1}, v_i)\) (or \((v_k, v_1)\) if \( i = 1 \)) is a back edge.
**Definition**

A **partially ordered set** is a set $S$ along with a binary relation $\preceq$ such that $\preceq$ is

1. **reflexive** ($a \preceq a$ for all $a \in V$),
2. **anti-symmetric** ($a \preceq b$ and $a \neq b$ implies $b \not\preceq a$), and
3. **transitive** ($a \preceq b$ and $b \preceq c$ implies $a \preceq c$).

**Example:** For numbers in the plane define $(x, y) \preceq (x', y')$ iff $x \leq x'$ and $y \leq y'$.

**Observation:** A finite partially ordered set is equivalent to a **DAG**. (No equal elements.)

**Observation:** A topological sort of a **DAG** corresponds to a complete (or total) ordering of the underlying partial order.
Definition

A partially ordered set is a set $S$ along with a binary relation $\preceq$ such that $\preceq$ is

1. reflexive ($a \preceq a$ for all $a \in V$),
2. anti-symmetric ($a \preceq b$ and $a \neq b$ implies $b \npreceq a$), and
3. transitive ($a \preceq b$ and $b \preceq c$ implies $a \preceq c$).

Example: For numbers in the plane define $(x, y) \preceq (x', y')$ iff $x \leq x'$ and $y \leq y'$.

Observation: A finite partially ordered set is equivalent to a DAG. (No equal elements.)

Observation: A topological sort of a DAG corresponds to a complete (or total) ordering of the underlying partial order.
A partially ordered set is a set $S$ along with a binary relation $\leq$ such that $\leq$ is

1. **reflexive** ($a \leq a$ for all $a \in V$),
2. **anti-symmetric** ($a \leq b$ and $a \neq b$ implies $b \not\leq a$), and
3. **transitive** ($a \leq b$ and $b \leq c$ implies $a \leq c$).

**Example:** For numbers in the plane define $(x, y) \leq (x', y')$ iff $x \leq x'$ and $y \leq y'$.

**Observation:** A finite partially ordered set is equivalent to a DAG. (No equal elements.)

**Observation:** A topological sort of a DAG corresponds to a complete (or total) ordering of the underlying partial order.
Part II

Linear time algorithm for finding all strong connected components of a directed graph
Finding all SCCs of a Directed Graph

Problem

Given a directed graph \( G = (V, E) \), output all its strong connected components.

Straightforward algorithm:

For each vertex \( u \in V \) do

find \( SCC(G, u) \) the strong component containing \( u \) as follows:

- Obtain \( rch(G, u) \) using \( DFS(G, u) \)
- Obtain \( rch(G^{rev}, u) \) using \( DFS(G^{rev}, u) \)
- Output \( SCC(G, u) = rch(G, u) \cap rch(G^{rev}, u) \)

Running time: \( O(n(n + m)) \)

Is there an \( O(n + m) \) time algorithm?
Finding all SCCs of a Directed Graph

Problem
Given a directed graph $G = (V, E)$, output all its strong connected components.

Straightforward algorithm:
For each vertex $u \in V$ do

1. find $SCC(G, u)$ the strong component containing $u$ as follows:
   - Obtain $rch(G, u)$ using $DFS(G, u)$
   - Obtain $rch(G^{rev}, u)$ using $DFS(G^{rev}, u)$
   - Output $SCC(G, u) = rch(G, u) \cap rch(G^{rev}, u)$

Running time: $O(n(n + m))$

Is there an $O(n + m)$ time algorithm?
Finding all SCCs of a Directed Graph

**Problem**

Given a directed graph \( G = (V, E) \), output all its strong connected components.

**Straightforward algorithm:**

For each vertex \( u \in V \) do

1. find \( SCC(G, u) \) the strong component containing \( u \) as follows:
   - Obtain \( rch(G, u) \) using \( DFS(G, u) \)
   - Obtain \( rch(G^{rev}, u) \) using \( DFS(G^{rev}, u) \)
   - Output \( SCC(G, u) = rch(G, u) \cap rch(G^{rev}, u) \)

**Running time:** \( O(n(n + m)) \)

**Is there an \( O(n + m) \) time algorithm?**
Proposition

For a directed graph $G$, its meta-graph $G^{\text{SCC}}$ is a DAG.
Linear-time Algorithm for SCCs: Ideas

Exploit structure of meta-graph.

### Algorithm

- Let $u$ be a vertex in a sink SCC of $G^{SCC}$
- Do $\text{DFS}(u)$ to compute $\text{SCC}(u)$
- Remove $\text{SCC}(u)$ and repeat

### Justification

- $\text{DFS}(u)$ only visits vertices (and edges) in $\text{SCC}(u)$
- $\text{DFS}(u)$ takes time proportional to size of $\text{SCC}(u)$
- Therefore, total time $O(n + m)$!
How do we find a vertex in the sink SCC of $G^{SCC}$?

Can we obtain an *implicit* topological sort of $G^{SCC}$ without computing $G^{SCC}$?

Answer: DFS($G$) gives some information!
Big Challenge(s)

How do we find a vertex in the sink SCC of $G^{\text{SCC}}$?

Can we obtain an *implicit* topological sort of $G^{\text{SCC}}$ without computing $G^{\text{SCC}}$?

Answer: $\text{DFS}(G)$ gives some information!
Big Challenge(s)

How do we find a vertex in the sink SCC of $G^{SCC}$?

Can we obtain an *implicit* topological sort of $G^{SCC}$ without computing $G^{SCC}$?

Answer: $\text{DFS}(G)$ gives some information!
Post-visit times of SCCs

Definition

Given $G$ and a SCC $S$ of $G$, define $\text{post}(S) = \max_{u \in S} \text{post}(u)$ where $\text{post}$ numbers are with respect to some $\text{DFS}(G)$. 
An Example

Figure: Graph $G$

Figure: Graph with pre-post times for $\text{DFS}(A)$; black edges in tree

Figure: $G^{\text{SCC}}$ with post times
If $S$ and $S'$ are SCCs in $G$ and $(S, S')$ is an edge in $G^{SCC}$ then $\text{post}(S) > \text{post}(S')$.

Proof.
Let $u$ be first vertex in $S \cup S'$ that is visited.
- If $u \in S$ then all of $S'$ will be explored before $\text{DFS}(u)$ completes.
- If $u \in S'$ then all of $S'$ will be explored before any of $S$.

A False Statement: If $S$ and $S'$ are SCCs in $G$ and $(S, S')$ is an edge in $G^{SCC}$ then for every $u \in S$ and $u' \in S'$, $\text{post}(u) > \text{post}(u')$. 
Proposition

If $S$ and $S'$ are SCCs in $G$ and $(S, S')$ is an edge in $G^{SCC}$ then $\text{post}(S) > \text{post}(S')$.

Proof.

Let $u$ be first vertex in $S \cup S'$ that is visited.

- If $u \in S$ then all of $S'$ will be explored before $\text{DFS}(u)$ completes.
- If $u \in S'$ then all of $S'$ will be explored before any of $S$.

A False Statement: If $S$ and $S'$ are SCCs in $G$ and $(S, S')$ is an edge in $G^{SCC}$ then for every $u \in S$ and $u' \in S'$, $\text{post}(u) > \text{post}(u')$. 
Corollary

Ordering SCCs in decreasing order of $\text{post}(S)$ gives a topological ordering of $G^{\text{SCC}}$.

Recall: for a DAG, ordering nodes in decreasing post-visit order gives a topological sort.

So...

$\text{DFS}(G)$ gives some information on topological ordering of $G^{\text{SCC}}$!
Corollary

Ordering $\text{SCC}$s in decreasing order of $\text{post}(S)$ gives a topological ordering of $G^{\text{SCC}}$.

Recall: for a DAG, ordering nodes in decreasing post-visit order gives a topological sort.

So...

$\text{DFS}(G)$ gives some information on topological ordering of $G^{\text{SCC}}$!
Proposition

The vertex $u$ with the highest post visit time belongs to a source $SCC$ in $G^{SCC}$.

Proof.

- $post(SCC(u)) = post(u)$
- Thus, $post(SCC(u))$ is highest and will be output first in topological ordering of $G^{SCC}$. 
Proposition

The vertex $u$ with the highest post visit time belongs to a source SCC in $G^{SCC}$.

Proof.

- $\text{post}(\text{SCC}(u)) = \text{post}(u)$
- Thus, $\text{post}(\text{SCC}(u))$ is highest and will be output first in topological ordering of $G^{SCC}$. 
Proposition

The vertex $u$ with highest post visit time in $\text{DFS}(G^{rev})$ belongs to a sink SCC of $G$.

Proof.

- $u$ belongs to source SCC of $G^{rev}$
- Since graph of SCCs of $G^{rev}$ is the reverse of $G^{SCC}$, $\text{SCC}(u)$ is sink SCC of $G$. 
Finding Sinks

**Proposition**

The vertex $u$ with highest post visit time in $\text{DFS}(G^{rev})$ belongs to a sink SCC of $G$.

**Proof.**

- $u$ belongs to source SCC of $G^{rev}$
- Since graph of SCCs of $G^{rev}$ is the reverse of $G^{SCC}$, $\text{SCC}(u)$ is sink SCC of $G$. 
Linear Time Algorithm

...for computing the strong connected components in $G$

Do $\text{DFS}(G^\text{rev})$ and sort vertices in decreasing post order.
Mark all nodes as unvisited
for each $u$ in the computed order do
  if $u$ is not visited then
    $\text{DFS}(u)$
    Let $S_u$ be the nodes reached by $u$
    Output $S_u$ as a strong connected component
    Remove $S_u$ from $G$

Analysis

Running time is $O(n + m)$. (Exercise)
Graph $G$: 

$$
\begin{array}{cccc}
B & A & C \\
E & F & D \\
G & H \\
\end{array}
$$

$\Rightarrow$

Reverse graph $G^{\text{rev}}$: 

$$
\begin{array}{cccc}
B & A & C \\
E & F & D \\
G & H \\
\end{array}
$$

DFS of reverse graph: 

Pre/Post DFS numbering of reverse graph:

$$
\begin{array}{cccc}
B & A & C \\
E & F & D \\
G & H \\
\end{array}
$$

$$
\begin{array}{cccc}
[7, 12] & [1, 6] \\
[9, 10] & \ [8, 11] \\
[13, 16] & [14, 15] \\
\end{array}
$$
Linear Time Algorithm: An Example

Removing connected components: 1

Original graph $G$ with rev post numbers:

Do DFS from vertex $G$ remove it.

$\text{SCC computed: } \{G\}$
Linear Time Algorithm: An Example
Removing connected components: 2

Do **DFS** from vertex **G**
remove it.

SCC computed:
{G}

Do **DFS** from vertex **H**,
remove it.

SCC computed:
{G}, {H}
Linear Time Algorithm: An Example

Removing connected components: 3

Do **DFS** from vertex **H**, remove it.

Do **DFS** from vertex **F**
Remove visited vertices: \{F, B, E\}.

**SCC** computed:
\{G\}, \{H\}

**SCC** computed:
\{G\}, \{H\}, \{F, B, E\}
Linear Time Algorithm: An Example
Removing connected components: 4

Do **DFS** from vertex **F**
Remove visited vertices: \{F, B, E\}.

Do **DFS** from vertex **A**
Remove visited vertices: \{A, C, D\}.

**SCC** computed:
\{G\}, \{H\}, \{F, B, E\}

**SCC** computed:
\{G\}, \{H\}, \{F, B, E\}, \{A, C, D\}
SCC computed:
\{G\}, \{H\}, \{F, B, E\}, \{A, C, D\}
Which is the correct answer!
Obtaining the meta-graph from strong connected components

**Exercise:** Given all the strong connected components of a directed graph $G = (V, E)$ show that the meta-graph $G_{SCC}$ can be obtained in $O(m + n)$ time.
Correctness: more details

- Let $S_1, S_2, \ldots, S_k$ be strong components in $G$.
- Strong components of $G^{rev}$ and $G$ are same and meta-graph of $G$ is reverse of meta-graph of $G^{rev}$.
- Consider $\text{DFS}(G^{rev})$ and let $u_1, u_2, \ldots, u_k$ be such that $\text{post}(u_i) = \text{post}(S_i) = \max_{v \in S_i} \text{post}(v)$.
- Assume without loss of generality that $\text{post}(u_k) > \text{post}(u_{k-1}) \geq \ldots \geq \text{post}(u_1)$ (renumber otherwise). Then $S_k, S_{k-1}, \ldots, S_1$ is a topological sort of meta-graph of $G^{rev}$ and hence $S_1, S_2, \ldots, S_k$ is a topological sort of the meta-graph of $G$.
- $u_k$ has highest post number and $\text{DFS}(u_k)$ will explore all of $S_k$ which is a sink component in $G$.
- After $S_k$ is removed $u_{k-1}$ has highest post number and $\text{DFS}(u_{k-1})$ will explore all of $S_{k-1}$ which is a sink component in remaining graph $G - S_k$. Formal proof by induction.
Correctness: more details

- let $S_1, S_2, \ldots, S_k$ be strong components in $G$
- Strong components of $G^\text{rev}$ and $G$ are same and meta-graph of $G$ is reverse of meta-graph of $G^\text{rev}$.
- consider $\text{DFS}(G^\text{rev})$ and let $u_1, u_2, \ldots, u_k$ be such that $\text{post}(u_i) = \text{post}(S_i) = \max_{v \in S_i} \text{post}(v)$.
- Assume without loss of generality that $\text{post}(u_k) > \text{post}(u_{k-1}) \geq \ldots \geq \text{post}(u_1)$ (renumber otherwise). Then $S_k, S_{k-1}, \ldots, S_1$ is a topological sort of meta-graph of $G^\text{rev}$ and hence $S_1, S_2, \ldots, S_k$ is a topological sort of the meta-graph of $G$.
- $u_k$ has highest post number and $\text{DFS}(u_k)$ will explore all of $S_k$ which is a sink component in $G$.
- After $S_k$ is removed $u_{k-1}$ has highest post number and $\text{DFS}(u_{k-1})$ will explore all of $S_{k-1}$ which is a sink component in remaining graph $G - S_k$. Formal proof by induction.
Let $S_1, S_2, \ldots, S_k$ be strong components in $G$.

Strong components of $G^{\text{rev}}$ and $G$ are same and meta-graph of $G$ is reverse of meta-graph of $G^{\text{rev}}$.

Consider $\text{DFS}(G^{\text{rev}})$ and let $u_1, u_2, \ldots, u_k$ be such that $\text{post}(u_i) = \text{post}(S_i) = \max_{v \in S_i} \text{post}(v)$.

Assume without loss of generality that $\text{post}(u_k) > \text{post}(u_{k-1}) \geq \ldots \geq \text{post}(u_1)$ (renumber otherwise). Then $S_k, S_{k-1}, \ldots, S_1$ is a topological sort of meta-graph of $G^{\text{rev}}$ and hence $S_1, S_2, \ldots, S_k$ is a topological sort of the meta-graph of $G$.

$u_k$ has highest post number and $\text{DFS}(u_k)$ will explore all of $S_k$ which is a sink component in $G$.

After $S_k$ is removed $u_{k-1}$ has highest post number and $\text{DFS}(u_{k-1})$ will explore all of $S_{k-1}$ which is a sink component in remaining graph $G - S_k$. Formal proof by induction.
Correctness: more details

- let $S_1, S_2, \ldots, S_k$ be strong components in $G$
- Strong components of $G^{\text{rev}}$ and $G$ are same and meta-graph of $G$ is reverse of meta-graph of $G^{\text{rev}}$.
- consider $\text{DFS}(G^{\text{rev}})$ and let $u_1, u_2, \ldots, u_k$ be such that $\text{post}(u_i) = \text{post}(S_i) = \max_{v \in S_i} \text{post}(v)$.
- Assume without loss of generality that $\text{post}(u_k) > \text{post}(u_{k-1}) \geq \ldots \geq \text{post}(u_1)$ (renumber otherwise). Then $S_k, S_{k-1}, \ldots, S_1$ is a topological sort of meta-graph of $G^{\text{rev}}$ and hence $S_1, S_2, \ldots, S_k$ is a topological sort of the meta-graph of $G$.
- $u_k$ has highest post number and $\text{DFS}(u_k)$ will explore all of $S_k$ which is a sink component in $G$.
- After $S_k$ is removed $u_{k-1}$ has highest post number and $\text{DFS}(u_{k-1})$ will explore all of $S_{k-1}$ which is a sink component in remaining graph $G - S_k$. Formal proof by induction.
Correctness: more details

- Let $S_1, S_2, \ldots, S_k$ be strong components in $G$.
- Strong components of $G^{rev}$ and $G$ are same and meta-graph of $G$ is reverse of meta-graph of $G^{rev}$.
- Consider $\text{DFS}(G^{rev})$ and let $u_1, u_2, \ldots, u_k$ be such that $\text{post}(u_i) = \text{post}(S_i) = \max_{v \in S_i} \text{post}(v)$.
- Assume without loss of generality that $\text{post}(u_k) > \text{post}(u_{k-1}) \geq \ldots \geq \text{post}(u_1)$ (renumber otherwise). Then $S_k, S_{k-1}, \ldots, S_1$ is a topological sort of meta-graph of $G^{rev}$ and hence $S_1, S_2, \ldots, S_k$ is a topological sort of the meta-graph of $G$.
- $u_k$ has highest post number and $\text{DFS}(u_k)$ will explore all of $S_k$ which is a sink component in $G$.
- After $S_k$ is removed $u_{k-1}$ has highest post number and $\text{DFS}(u_{k-1})$ will explore all of $S_{k-1}$ which is a sink component in remaining graph $G - S_k$. Formal proof by induction.
Correctness: more details

- let $S_1, S_2, \ldots, S_k$ be strong components in $G$
- Strong components of $G^{rev}$ and $G$ are same and meta-graph of $G$ is reverse of meta-graph of $G^{rev}$.
- consider $DFS(G^{rev})$ and let $u_1, u_2, \ldots, u_k$ be such that $post(u_i) = post(S_i) = \max_{v \in S_i} post(v)$.
- Assume without loss of generality that $post(u_k) > post(u_{k-1}) \geq \ldots \geq post(u_1)$ (renumber otherwise). Then $S_k, S_{k-1}, \ldots, S_1$ is a topological sort of meta-graph of $G^{rev}$ and hence $S_1, S_2, \ldots, S_k$ is a topological sort of the meta-graph of $G$.
- $u_k$ has highest post number and $DFS(u_k)$ will explore all of $S_k$ which is a sink component in $G$.
- After $S_k$ is removed $u_{k-1}$ has highest post number and $DFS(u_{k-1})$ will explore all of $S_{k-1}$ which is a sink component in remaining graph $G - S_k$. Formal proof by induction.
Part III

An Application to make
Unix utility for automatically building large software applications

A makefile specifies
- Object files to be created,
- Source/object files to be used in creation, and
- How to create them
An Example makefile

project:  main.o utils.o command.o
        cc -o project main.o utils.o command.o

main.o:  main.c defs.h
        cc -c main.c
utils.o: utils.c defs.h command.h
        cc -c utils.c
command.o: command.c defs.h command.h
        cc -c command.c
makefile as a Digraph

- main.c
- utils.c
- defs.h
- command.h
- command.c

main.o
utils.o
project
command.o

main.o

Sariel (UIUC)
CS473 43
Fall 2011 43 / 50
Computational Problems for make

- Is the makefile reasonable?
- If it is reasonable, in what order should the object files be created?
- If it is not reasonable, provide helpful debugging information.
- If some file is modified, find the fewest compilations needed to make application consistent.
Is the makefile reasonable? Is $G$ a DAG?

If it is reasonable, in what order should the object files be created? Find a topological sort of a DAG.

If it is not reasonable, provide helpful debugging information. Output a cycle. More generally, output all strong connected components.

If some file is modified, find the fewest compilations needed to make application consistent.

- Find all vertices reachable (using DFS/BFS) from modified files in directed graph, and recompile them in proper order. Verify that one can find the files to recompile and the ordering in linear time.
Take away Points

- Given a directed graph $G$, its $\text{SCCs}$ and the associated acyclic meta-graph $G^{\text{SCC}}$ give a structural decomposition of $G$ that should be kept in mind.

- There is a $\text{DFS}$ based linear time algorithm to compute all the $\text{SCCs}$ and the meta-graph. Properties of $\text{DFS}$ crucial for the algorithm.

- $\text{DAGs}$ arise in many application and topological sort is a key property in algorithm design. Linear time algorithms to compute a topological sort (there can be many possible orderings so not unique).
Assume \((u, v) \in E(G)\) is not possible.

If \(\text{pre}(v) > \text{pre}(u)\) then \(u \Rightarrow v\) is a cycle.

If \(\text{pre}(v) < \text{pre}(u)\) then \(v \Rightarrow u\) is a cycle.

Contradiction.