CS 473: Algorithms

Chandra Chekuri
chekuri@cs.illinois.edu
3228 Siebel Center

University of Illinois, Urbana-Champaign

Fall 2010
Strong Connected Components (SCCs)

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

- 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 $G$. 

**Figure:** Graph $G$

**Figure:** Graph of SCCs $G^{\text{SCC}}$
Reversal and SCCs

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.
Proposition

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

Proof.

If $G^{\text{SCC}}$ 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 $G$. 
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.
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.
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.
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.
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 if each node is in its own strong connected component.

Formal proofs: exercise.
A topological ordering/sorting of $G = (V, E)$ is an ordering $<_{\text{on}}$ 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.
Lemma

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

Proof.

Only if: 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 $u_1 < u_2 < \ldots < u_k < u_1$! A contradiction.
Lemma

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

Proof.

If: 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:
Topological Sort: An Example

Output: 1
Topological Sort: An Example

Output: 1 2
Topological Sort: An Example

Output: 1 2 3
Topological Sort: An Example

Output: 1 2 3 4
Topological Sort: Another Example

a → b → c → d → e → f → g → h

a d b c e g f h
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?

**Question:** What is a DAG with the least number of distinct topological sorts for a given number $n$ of vertices?
DFS to check for Acyclicity and Topological Ordering

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

DFS based algorithm:
- Compute 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 DFS($G$).
- Proposition If $G$ is a DAG and $\text{post}(v) > \text{post}(u)$, then $(u, v)$ is not in $G$. 
DFS to check for Acyclicity and Topological Ordering

Question

Given \( G \), is it a DAG? If it is, generate a topological sort.

DFS based algorithm:

- Compute 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 DFS(\( G \)).
- Proposition If \( G \) is a DAG and post(\( v \)) > post(\( u \)), then (\( u, v \)) is not in \( G \).
DFS to check for Acyclicity and 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

\begin{itemize}
\item \textbf{Graph:}
\begin{itemize}
\item Node 1
\item Node 2 connected to Node 3
\item Node 3 connected to Node 4
\end{itemize}
\end{itemize}

\begin{itemize}
\item \textbf{Tree:}
\begin{itemize}
\item Node 2 with children 3, 4
\item Node 3 with child 4
\item Node 4 with child 2
\end{itemize}
\end{itemize}
Proposition

$G$ has a cycle iff there is a back-edge in $DFS(G)$. 

Proof.

If: $(u, v)$ is a back edge implies there is a cycle $C$ consisting of the path from $v$ to $u$ in 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 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 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 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 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

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

Proof.

Assume $\text{post}(v) > \text{post}(u)$ and $(u, v)$ is an edge in $G$. We derive a contradiction. One of two cases holds from DFS property.

- **Case 1:** $[\text{pre}(u), \text{post}(u)]$ is contained in $[\text{pre}(v), \text{post}(v)]$. Implies that $(u, v)$ is a back edge but a DAG has no back edges!

- **Case 2:** $[\text{pre}(u), \text{post}(u)]$ is disjoint from $[\text{pre}(v), \text{post}(v)]$. This cannot happen since $v$ would be explored from $u$. 
A partially ordered set is a set $S$ along with a binary relation $\preceq$ such that $\preceq$ is (i) reflexive ($a \preceq a$ for all $a \in V$), (ii) anti-symmetric ($a \preceq b$ implies $b \not\preceq a$) and (iii) 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.

Observation: A topological sort of a DAG corresponds to a complete (or total) ordering of the underlying partial order.
DAGs and Partial Orders

Definition

A partially ordered set is a set $S$ along with a binary relation $\leq$ such that $\leq$ is (i) reflexive ($a \leq a$ for all $a \in V$), (ii) anti-symmetric ($a \leq b$ implies $b \not\leq a$) and (iii) 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 $y \leq y'$.
A partially ordered set is a set $S$ along with a binary relation $\preceq$ such that $\preceq$ is (i) reflexive ($a \preceq a$ for all $a \in V$), (ii) anti-symmetric ($a \preceq b$ implies $b \not\preceq a$) and (iii) 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.

**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 \( rch(G^{\text{rev}}, u) \) using DFS \((G^{\text{rev}}, u)\)

Output \( SCC(G, u) = rch(G, u) \cap rch(G^{\text{rev}}, u) \)

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

- 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:** \( O(n(n + m)) \)
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?
Structure of a Directed Graph

Figure: Graph $G$

![Graph $G$](image)

Figure: Graph of SCCs $G^{SCC}$

![Graph of SCCs $G^{SCC}$](image)

Proposition

For a directed graph $G$, its meta-graph $G^{SCC}$ 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 DFS($u$) to compute SCC($u$)
- Remove SCC($u$) and repeat

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

![Graph $G$](image)

Figure: Graph with pre-post times for DFS(A); black edges in tree

![Graph with pre-post times](image)

Figure: $G^{SCC}$ with post times

![$G^{SCC}$ with post times](image)
Proposition

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

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

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

Figure: Graph $G$

Figure: Graph with pre-post times for DFS(A); black edges in tree

Figure: $G^{SCC}$ with post times
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 DFS\( (u) \) to compute SCC\( (u) \)
- Remove SCC\( (u) \) and repeat

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

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

Proof.

Thus, post($SCC(u)$) is highest and will be output first in topological ordering of $G^{SCC}$. 
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 $\text{DFS}(\overline{G})$ belongs to a sink SCC of $G$. 
Finding Sinks

Proposition

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

Proof.

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

Linear Time Algorithm

Do DFS($G^{rev}$) 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
    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)
Linear Time Algorithm: An Example

Figure: Graph $G$

**Figure:** Graph $G$
Linear Time Algorithm: An Example

Figure: Graph $G$

Figure: $G^\text{rev}$
Linear Time Algorithm: An Example

Figure: Graph $G$

Figure: $G^{\text{rev}}$ with pre-post times. Red edges not traversed in DFS.
Linear Time Algorithm: An Example

**Figure:** Graph $G$

**Figure:** $G^\text{rev}$ with pre-post times. Red edges not traversed in DFS

Order of second DFS: $\text{DFS}(G) = \{G\}$;
Linear Time Algorithm: An Example

Figure: Graph $G$

Order of second DFS: $\text{DFS}(G) = \{G\}$; $\text{DFS}(H) = \{H\}$;

Figure: $G^{\text{rev}}$ with pre-post times. Red edges not traversed in DFS

Chekuri CS473
Linear Time Algorithm: An Example

Figure: Graph $G$

Order of second DFS: $\text{DFS}(G) = \{ G \}; \text{DFS}(H) = \{ H \};$
$\text{DFS}(B) = \{ B, E, F \};$

Figure: $G^{\text{rev}}$ with pre-post times. Red edges not traversed in DFS

Chekuri
CS473
Linear Time Algorithm: An Example

**Figure:** Graph $G$

**Figure:** $G^{rev}$ with pre-post times. Red edges not traversed in DFS

Order of second DFS: $\text{DFS}(G) = \{G\}$; $\text{DFS}(H) = \{H\}$; $\text{DFS}(B) = \{B, E, F\}$; $\text{DFS}(A) = \{A, C, D\}$. 
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 $G$
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 $G^{rev}$. 
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 DFG($G^{rev}$) 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 $G^{rev}$.

$u_k$ 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.
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 DFG($G^{rev}$) 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 $G$. 

\[ \text{Chekuri CS473} \]
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 DFG($G^{rev}$) 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 DFS($u_k$) will explore all of $S_k$ which is a sink component in $G$. 

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 DFG($G^{rev}$) 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 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
make Utility [Feldman]

- Unix utility for automatically building large software applications
Unix utility for automatically building large software applications
A makefile specifies
**make Utility [Feldman]**

- Unix utility for automatically building large software applications
- A makefile specifies
  - Object files to be created,
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
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
- command.o
- project
Computational Problems for make

- Is the makefile reasonable?
Computational Problems for make

- Is the makefile reasonable?
- If it is reasonable, in what order should the object files be created?
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.
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.
Algorithms for make

- Is the makefile reasonable? Is $G$ a DAG?
Algorithms for make

- 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.
Algorithms for make

- 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.
Algorithms for *make*

- 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.

Verifying the cycle and the ordering in linear time.
Algorithms for make

- 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.
Takeaway Points

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

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

- 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).