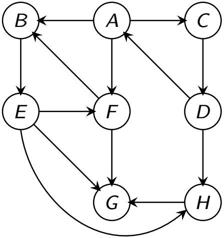
CS 473: Algorithms

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

University of Illinois, Urbana-Champaign

Fall 2009

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.

Graph of SCCs

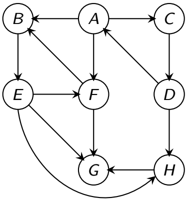


Figure: Graph G

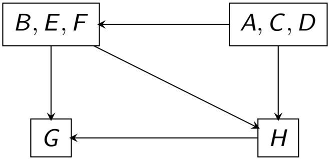


Figure: Graph of SCCs G^{SCC}

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.

Reversal and SCCs

Proposition

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

Proof.

Exercise. □

SCCs and DAGs

Proposition

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

SCCs and DAGs

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. $\square$

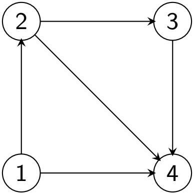
Part I

Directed Acyclic Graphs

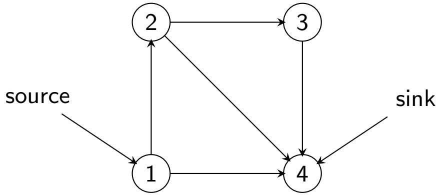
Directed Acyclic Graphs

Definition

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



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.

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.

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 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 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 component.

Formal proofs: exercise.

Topological Ordering/Sorting

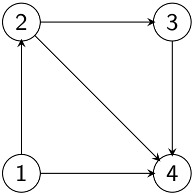


Figure: Graph G

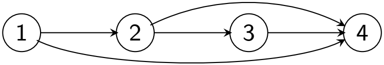


Figure: Topological Ordering of G

Definition

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

DAGs and Topological Sort

Lemma

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

DAGs and Topological Sort

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 $<$.
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. $\square$

DAGs and Topological Sort

Lemma

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

Proof.

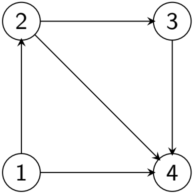
If: Consider the following algorithm:

- 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.

□

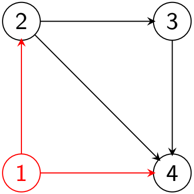
Exercise: show above algorithm can be implemented in $O(m + n)$ time.

Topological Sort: An Example



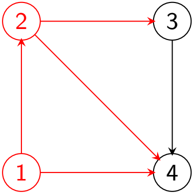
Output:

Topological Sort: An Example



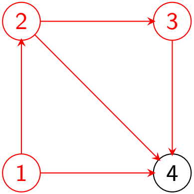
Output: 1

Topological Sort: An Example



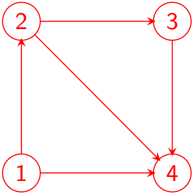
Output: 1 2

Topological Sort: An Example



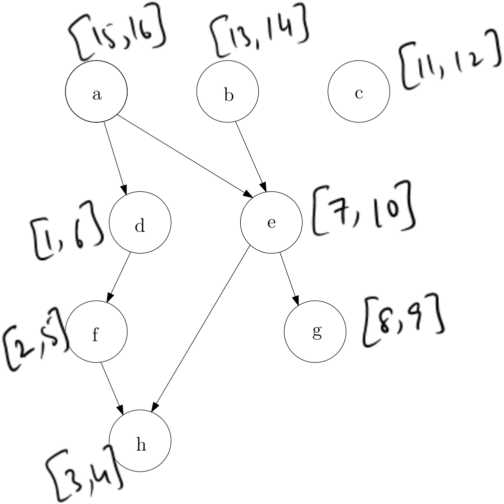
Output: 1 2 3

Topological Sort: An Example



Output: 1 2 3 4

Topological Sort: Another Example



a d b c e g f h

a b c e g d 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 Acylicity and Topological Ordering

Question

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

DFS to check for Acylicity 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.

DFS to check for Acylicity 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:

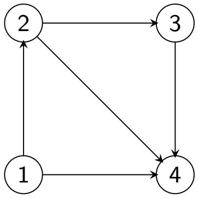
Proposition

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.

Example



Back edge and Cycles

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).

Back edge and Cycles

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

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

Proof.

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

- Case 1: $[pre(u), post(u)]$ is contained in $[pre(v), post(v)]$. Implies that (u, v) is a back edge but a DAG has no back edges!
- Case 2: $[pre(u), post(u)]$ is disjoint from $[pre(v), post(v)]$. This cannot happen since v would be explored from u.

□

DAGs and Partial Orders

Definition

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 $a \preceq c$).

DAGs and Partial Orders

Definition

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 $y \leq y'$.

DAGs and Partial Orders

Definition

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

Finding all SCCs of a Graph

Problem

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

Finding all SCCs of a Graph

Problem

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

Algorithm from previous lecture:

For each vertex $u \in V$ do
    find $SCC(G, u)$ the strong component containing u as follows:
        Obtain $\mathtt{rch}(G, u)$ using $DFS(G, u)$
        Obtain $\mathtt{rch}(G^{\mathrm{rev}}, u)$ using $DFS(G^{\mathrm{rev}}, u)$
        Output $SCC(G, u) = \mathtt{rch}(G, u) \cap \mathtt{rch}(G^{\mathrm{rev}}, u)$

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

Finding all SCCs of a Graph

Problem

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

Algorithm from previous lecture:

```
For each vertex u ∈ 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) ∩ rch(G^rev, u)
```

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

Is there an $O(n + m)$ time algorithm?

Structure of a Directed Graph

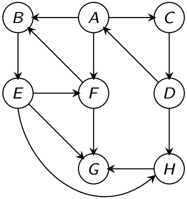


Figure: Graph G

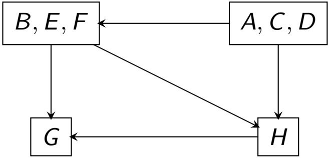


Figure: Graph of SCCs G^{SCC}

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 $\texttt{DFS}(u)$ to compute $\mathrm{SCC}(u)$
- Remove $\mathrm{SCC}(u)$ and repeat

Justification

- $\texttt{DFS}(u)$ only visits vertices (and edges) in $\mathrm{SCC}(u)$
- $\texttt{DFS}(u)$ takes time proportional to size of $\mathrm{SCC}(u)$
- Therefore, total time $O(n+m)$!

Big Challenge(s)

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!

Post-visit times of SCCs

Definition

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

An Example

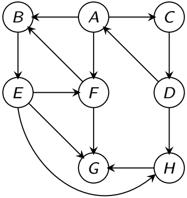


Figure: Graph G

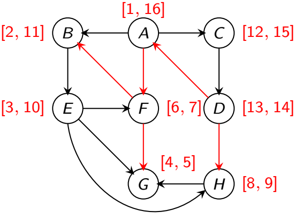


Figure: Graph with pre-post times for DFS(A); black edges in tree

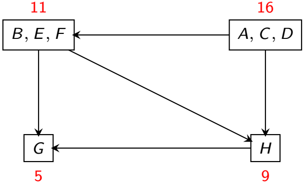


Figure: G^{SCC} with post times

(A,C,D) (B,E,F)
(H) (G)

G^{SCC} and post-visit times

Proposition

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

G^{SCC} and post-visit times

Proposition

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

Proof.

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

$\square$

G^{SCC} and post-visit times

Proposition

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

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 $\mathtt{DFS}(u)$ completes.

$\square$

Proposition

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

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

$\square$

G^{SCC} and post-visit times

Proposition

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

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

$\square$

G^{SCC} and post-visit times

Proposition

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

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

$\square$

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 $u' \in S'$, $\mathrm{post}(u) > \mathrm{post}(u')$.

Topological ordering of G^{SCC}

Corollary

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

Topological ordering of G^{SCC}

Corollary

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

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

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

An Example

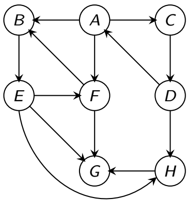


Figure: Graph G

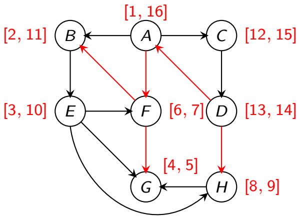


Figure: Graph with pre-post times for DFS(A); black edges in tree

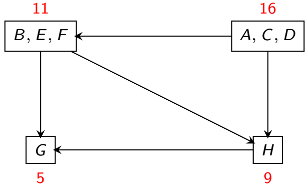


Figure: G^{SCC} with post times

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 $\mathtt{DFS}(u)$ to compute $\mathrm{SCC}(u)$
- Remove $\mathrm{SCC}(u)$ and repeat

Justification

- $\mathtt{DFS}(u)$ only visits vertices (and edges) in $\mathrm{SCC}(u)$
- $\mathtt{DFS}(u)$ takes time proportional to size of $\mathrm{SCC}(u)$
- Therefore, total time $O(n + m)$!

Big Challenge(s)

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!

Finding Sources

Proposition

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

Finding Sources

Proposition

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

Proof.

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

$\square$

Finding Sinks

Proposition

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

Finding Sinks

Proposition

The vertex u with highest post visit time in $\texttt{DFS}(G^{\text{rev}})$ 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

```
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)
        Output all nodes reached by u as a strong component
        Remove these nodes from G
```

Analysis

Running time is $O(n+m)$.

Linear Time Algorithm: An Example

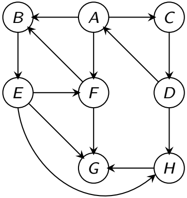


Figure: Graph G

Linear Time Algorithm: An Example

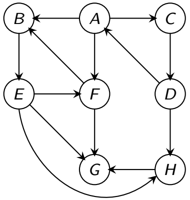


Figure: Graph G

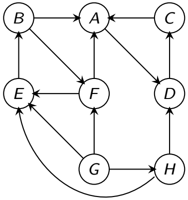


Figure: G^{rev}

Linear Time Algorithm: An Example

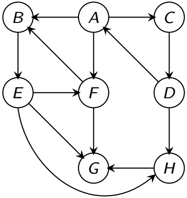


Figure: Graph G

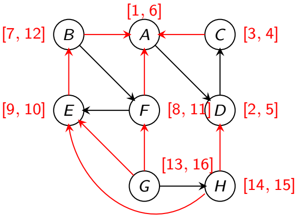


Figure: G^{rev} with pre-post times. Red edges not traversed in DFS

Linear Time Algorithm: An Example

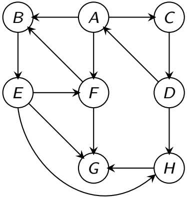


Figure: Graph G

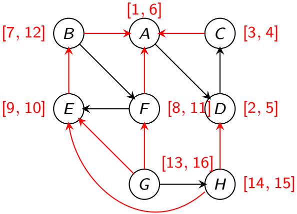


Figure: G^{rev} with pre-post times. Red edges not traversed in DFS

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

Linear Time Algorithm: An Example

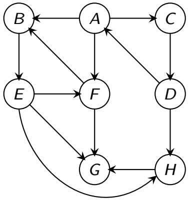


Figure: Graph G

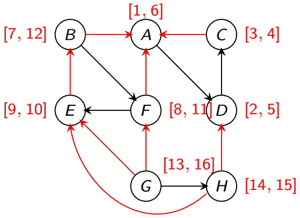


Figure: G^{rev} with pre-post times. Red edges not traversed in DFS

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

Linear Time Algorithm: An Example

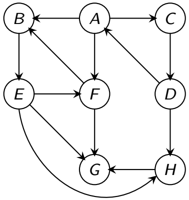


Figure: Graph G

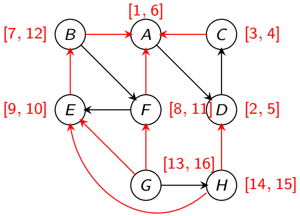


Figure: G^{rev} with pre-post times. Red edges not traversed in DFS

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

Linear Time Algorithm: An Example

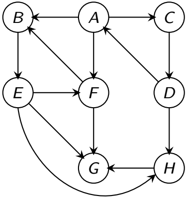


Figure: Graph G

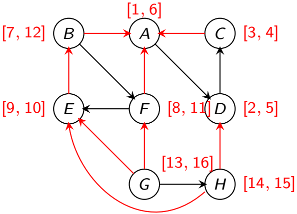


Figure: G^{rev} with pre-post times. Red edges not traversed in DFS

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

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 $\text{post}(u_i) = \text{post}(S_i) = \max_{v \in S_i} post(v)$.

Correctness: more details

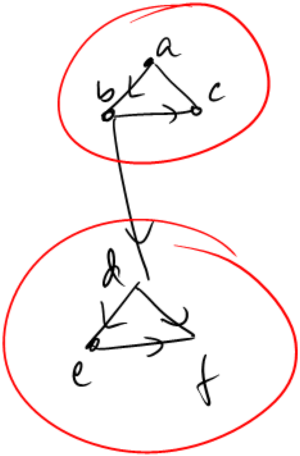
- let $S_1, S_2, \ldots, S_k$ be strong components in G
- Strong components of G^{rev} and G are same and meta-graph of G is reverse of meta-graph of G^{rev}.
- consider DFG(G^{rev}) and let $u_1, u_2, \ldots, u_k$ be such that $\text{post}(u_i) = \text{post}(S_i) = \max_{v \in S_i} post(v)$.
- Assume 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.

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 $\text{post}(u_i) = \text{post}(S_i) = \max_{v \in S_i} post(v)$.
- 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.

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 $\text{post}(u_i) = \text{post}(S_i) = \max_{v \in S_i} post(v)$.
- 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.


a
b
c
d
e
f

Part III

An Application to make

make Utility [Feldman]

- Unix utility for automatically building large software applications

make Utility [Feldman]

- 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,

make Utility [Feldman]

- 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

make Utility [Feldman]

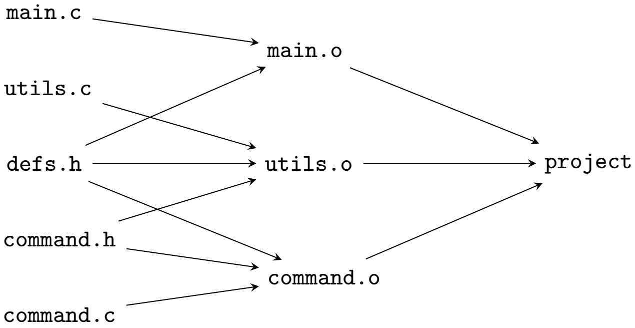
- Unix utility for automatically building large software applications
- A makefile specifies
  - Object files to be created,
  - Source/object files to be used in creation, and
  - How to create them

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



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.

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.

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.