More on DFS in Directed Graphs, and Strong Connected Components, and DAGs

Lecture 3
January 27, 2015

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:
1. Compute $\text{DFS}(G)$
2. If there is a back edge then G is not a DAG.
3. 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 $\text{post}(u) > \text{post}(v)$, then $(u \to v)$ is not in $G$.

Proof

Proposition
If $G$ is a DAG and $\text{post}(u) < \text{post}(v)$, 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.

1. **Case 1**: $[\text{pre}(u), \text{post}(u)]$ is contained in $[\text{pre}(v), \text{post}(v)]$.
2. $\implies u$ explored during $\text{DFS}(v)$.
3. $u$ descendant of $v$.
4. $(u, v) \in E(G)$ $\implies$ cycle in $G$ but $G$ is a DAG.

Proof continued

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

Proof continued...

Case 2: $[\text{pre}(u), \text{post}(u)]$ is disjoint from $[\text{pre}(v), \text{post}(v)]$.

1. By assumption: $\text{post}(u) < \text{post}(v)$.
2. $\implies \text{pre}(u) < \text{pre}(v)$.
3. $\text{DFS}$ visits $u$ first and then $v$.
4. If $(u \to v) \in E(G)$...
5. $\implies \text{DFS}$ explores $v$ during the $\text{DFS}$ of $u$.
6. $[\text{pre}(v), \text{post}(v)] \subseteq [\text{pre}(u), \text{post}(u)]$.
7. $\implies$ contradiction.
Example

Back edge and Cycles

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

Proof.
1. If: $(u, v)$ is a back edge $\implies$ there is a cycle $C$ in $G$:
   $C =$ path from $v$ to $u$ in $\text{DFS}$ tree + edge $(u \rightarrow v)$.
2. Only if: Suppose there is a cycle $C = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k \rightarrow v_1$.
   a. Let $v_i$ be first node in $C$ visited in $\text{DFS}$.
   b. All other nodes in $C$ are descendants of $v_i$ since they are reachable from $v_i$.
   c. Therefore, $(v_{i-1}, v_i)$ (or $(v_k, v_1)$ if $i = 1$) is a back edge.

Topological sorting of a DAG

Input: DAG $G$. With $n$ vertices and $m$ edges.

$O(n + m)$ algorithms for topological sorting

(A) Put source $s$ of $G$ as first in the order, remove $s$, and repeat.
   (Implementation not trivial.)
(B) Do $\text{DFS}$ of $G$.
   Compute post numbers.
   Sort vertices by decreasing post number.

Question
How to avoid sorting?
No need to sort - post numbering algorithm can output vertices...

DAGs and Partial Orders

Definition
A partially ordered set is a set $S$ along with a binary relation $\leq$ such that $\leq$ is
- reflexive ($a \leq a$ for all $a \in S$).
- anti-symmetric ($a \leq b$ and $a \neq b$ implies $b \not\leq a$), and
- 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.
What's **DAG** but a sweet old fashioned notion

Who needs a **DAG**...

### Example

- **V**: set of \( n \) products (say, \( n \) different types of tablets).
- Want to buy one of them, so you do market research...
- Online reviews compare only pairs of them. ...Not everything compared to everything.
- Given this partial information:
  - Decide what is the best product.
  - Decide what is the ordering of products from best to worst.
  - ...
Reminder II: Graph $G$ a vertex $F$ .. and its reachable set $rch(G, F)$

Graph $G$

Reachable set of vertices from $F$

Reminder III: Graph $G$ a vertex $F$ .. and the set of vertices that can reach it in $G$: $rch(G^{rev}, F)$

Graph $G$

Set of vertices that can reach $F$, computed via DFS in the reverse graph $G^{rev}$.

Reminder IV: Graph $G$ a vertex $F$ and...

its strong connected component in $G$: $SCC(G, F)$

Graph $G$

$rch(G, F)$

$rch(G^{rev}, F)$

$SCC(G, F) = rch(G, F) \cap rch(G^{rev}, F)$

Reminder II: Strong connected components (SCC)

Graph $G$

Graph of SCCs $G^{SCC}$

B, E, F

A, C, D

G

H
Finding all SCCs of a Directed Graph

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

Straightforward algorithm:
Mark all vertices in $V$ as not visited.
for each vertex $u \in V$ not visited yet do
find $\text{SCC}(G, u)$ the strong component of $u$:
Compute $\text{rch}(G, u)$ using $\text{DFS}(G, u)$
Compute $\text{rch}(G^{\text{rev}}, u)$ using $\text{DFS}(G^{\text{rev}}, u)$
$\text{SCC}(G, u) \leftarrow \text{rch}(G, u) \cap \text{rch}(G^{\text{rev}}, u)$
$\forall u \in \text{SCC}(G, u)$: Mark $u$ as visited.

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

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

Structure of a Directed Graph

Reminder
$G^{\text{SCC}}$ is created by collapsing every strong connected component to a single vertex.

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...

Wishful Thinking Algorithm
- Let $u$ be a vertex in a sink SCC of $G^{\text{SCC}}$
- 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 $\text{SCC}(u)$
- ... since there are no edges coming out a sink!
- $\text{DFS}(u)$ takes time proportional to size of $\text{SCC}(u)$
- Therefore, total time $O(n + m)$!

Big Challenge(s)

How do we find a vertex in a 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!
**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 post numbers are with respect to some $\text{DFS}(G)$.

**Graph of strong connected components**

... and post-visit times

**Proposition**
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.
1. If $u \in S$ then all of $S'$ will be explored before $\text{DFS}(u)$ completes.
2. 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 $u' \in S'$, $\text{post}(u) > \text{post}(u')$.

**An Example**

**Graph G**

**Graph with pre-post times for DFS(G); black edges in tree**

**Figure:** $G^{\text{SCC}}$ with post times

**Topological ordering of the strong components**

**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 $G^{\text{SCC}}$. 
Finding Sources

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

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

Finding Sinks

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

Proof.
1. $u$ belongs to source SCC of $G^{rev}$
2. 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 $\text{DFS}(G^{rev})$ 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)
Linear Time Algorithm: An Example

Removing connected components: 1

Original graph $G$ with rev post numbers:

- $G$
- $F$
- $E$
- $B$
- $C$
- $D$
- $H$
- $A$
- $16$
- $11$
- $6$
- $12$
- $10$
- $15$
- $5$
- $4$

$G 
FE
B C
D
H
A
16
11
612
10
15
5
4

⇒

Do DFS from vertex $G$, remove it.

SCC computed: 
\{G\}

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\}

Removing connected components: 3

Do DFS from vertex $H$, remove it.

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

Do DFS from vertex $B$, remove it.

Remove visited vertices: 
\{F, B, E\}

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

Do DFS from vertex $A$, remove it.

Remove visited vertices: 
\{A, C, D\}

SCC computed: 
\{G\}, \{H\}, \{F, B, E\}, \{A, C, D\}
Correctness: more details

1. Let $S_1, S_2, \ldots, S_k$ be strong components in $G$.
2. Strong components of $G^{rev}$ and $G$ are same and meta-graph of $G$ is reverse of meta-graph of $G^{rev}$.
3. Consider $\text{DFS}(G^{rev})$ and let $u_1, u_2, \ldots, u_k$ be such that
   $$\text{post}(u_i) = \text{post}(S_i) = \max_{v \in S_i} \text{post}(v).$$
4. 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 $G$.
5. $u_k$ has highest post number and $\text{DFS}(u_k)$ will explore all of $S_k$ which is a sink component in $G$.
6. 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.
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

```
main.c  main.o
       |
       v
utils.c
       |
       v
defs.h  utils.o
       |
       v
command.h
       |
       v
command.c
       |
       v
project
```

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?
- 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 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).

Part III

Not for lecture - why do we have to use the reverse graph in computing the SCC?

Finding a sink via post numbers in a DAG

Lemma

Let G be a DAG, and consider the vertex $u$ in G that minimizes post$(u)$. Then $u$ is a sink of G.

Proof.

The minimum post($) is assigned the first time DFS returns for its recursion. Let $\pi = v_1, v_2, \ldots, v_k = u$ be the sequence of vertices visited by the DFS at this point. Clearly, $u$ (i.e., $v_k$) can not have an edge going into $v_1,\ldots,v_{k-1}$ since this would violates the assumption that there are no cycles. Similarly, $u$ can not have an outgoing edge going into a vertex $z \in V(G) \setminus \{v_1,\ldots,v_k\}$, since the DFS would have continued into $z$, and $u$ would not have been the first vertex to get assigned a post number. We conclude that $u$ has no outgoing edges, and it is thus a sink.
Counterexample: Finding a source via min post numbers in a DAG

Counter example
Let $G$ be a DAG, and consider the vertex $u$ in $G$ that minimizes $\text{post}(u)$ is a source. This is FALSE.

the DFS numbering might be:
$A:[1,4]$
$B:[2,3]$
$C:[5,6]$

But clearly $B$ is not a source.

Finding a source via post numbers in a DAG

Lemma
Let $G$ be a DAG, and consider the vertex $u$ in $G$ that maximizes $\text{post}(u)$. Then $u$ is a source of $G$.

Proof: Exercise (And should already be in the slides.)

Meta graph computing the sink..

We proved:

Lemma
Consider the graph $G^{\text{SCC}}$, with every $CC$ $S \in V(G^{\text{SCC}})$ numbered by $\text{post}(S)$. Then:

$$\forall (S \rightarrow T) \in E(G^{\text{SCC}}) \quad \text{post}(S) > \text{post}(T).$$

- So, the SCC realizing $\min \text{post}(S)$ is indeed a sink of $G^{\text{SCC}}$.
- But how to compute this? Not clear at all.

Meta graph computing a source is easy!

- The SCC realizing $\max \text{post}(S)$ is a source of $G^{\text{SCC}}$.
- Furthermore, computing
  $$\max_{S \in V(G^{\text{SCC}})} \text{post}(S) = \max_{S \in V(G^{\text{SCC}})} \max_{v \in S} \text{post}(v) = \max_{v \in V(G)} \text{post}(v).$$
  is easy!
- So computing a source in the meta-graph is easy from the post numbering.
- But the algorithm needs a sink of the meta graph. Thus, we compute a vertex in the source SCC of the meta-graph of $(G^{\text{rev}})^{\text{SCC}} = (G^{\text{SCC}})^{\text{rev}}$. 