Depth first search: directed.

Directed graphs...with cycles.

Testing for cycle/Topological sort.

Back Edge: \((u,v)\) where \(\text{int}(v)\) contains \(\text{int}(u)\).

Thm: A graph has a cycle if and only if there is a back edge.

Proof Idea: Back edge \(\implies\) cycle!

Edge to ancestor, plus path from ancestor is cycle.

Interval of first explored vertex, \(v_0\), contains all others.

Edge from “last” vertex, \(v_0\), to \(v_0\) is back edge.

Topological Ordering: \(\pi\), where for all edges \((u,v)\), \(\pi(u) < \pi(v)\).

Thm: Reverse order of post number is a topological ordering.

Proof: No back edges!

Tree/Forward edge: \((u,v)\) where \(\text{int}(v)\) contains \(\text{int}(u)\).

Forward \((A,F)\): \([10,11]\) in \([0,13]\) or \([0,10,11,13]\).

Back edge \((u,v)\) where \(\text{int}(v)\) contains \(\text{int}(u)\).

\((C,B)\): \([3,4]\) in \([1,8]\) or \([1,3,4,8]\).

Cross edge \((u,v)\) where \(\text{int}(v)\) before \(\text{int}(u)\).

\((F,D)\): \([2,5]\) before \([10,11]\).

Topological Sort Example.

A linear order:

\[A,E,F,B,G,D,C\]

In DFS: When is A popped off stack?

plus Induction. \(\implies\) Reverse order of post numbering is topological ordering.

Lecture in a minute!

Quick Review:

DFS so far = how I learned to love the stack.

pre/post = time on stack.

Topological Ordering:

Inverse post ordering = topological ordering.

Remove source, repeat.

Strongly Connected Components: directed graphs.

Strong Connectivity for u and v.

On a cycle together.

Easy: \(O(|V||E|)\) algorithm.

Linear time algorithm!

Observation: Highest post in “source component”.

Find vertex in sink component.

Explore.

Repeat.

 DFS: so far.

How I learned to stop worrying
...and love the stack. See “Dr. Strangelove”.

\([\text{pre}(u),\text{post}(u)]\) is “clock interval on stack.”

\(v\) is an ancestor of \(u\)

Who is an ancestor of whom?

No cycles: no back edge

Back edge – edge to ancestor in “tree” of explore calls.”

Topological order - edge \((u,v)\) means \(u\) before \(v\)

Inverse post ordering is topological order.

Bonus Topological Order Algorithm:

remove source, repeat.
Topological Sort: DFS

- Last post order should...
  - (A) be first in linearization!
  - (B) be last in linearization!
  - (A). First!

Connectivity in Directed Graphs.

- Two nodes are connected...when?
  - When there is a path from \( u \) to \( v \)?
  - When there is a path from \( v \) to \( u \)?
  - Both!

- Nodes \( u \) and \( v \) are strongly connected
  - if there is a path from \( u \) to \( v \)
    and a path from \( v \) to \( u \).

  - Note: Nodes are strongly connected to themselves.
  - Path with zero edges in both directions!

Properties..

- Nodes \( u \) and \( v \) are strongly connected
  - if there is a path from \( u \) to \( v \)
  - and a path from \( v \) to \( u \).

  - Remember: Nodes are strongly connected to themselves.
  - True/False?
    - If \( u \) is strongly connected to \( v \) and \( v \) is strongly connected to \( w \)
      \( \Rightarrow u \) connected to \( w \).
    - True!
      - path from \( u \) (through \( v \)) to \( w \) and path from \( w \) (through \( v \)) to \( u \)
    - Transitive: \( u \) strongly connected to \( v \) strongly connected to \( w \)
      \( \Rightarrow u \) connected to \( w \).
    - Relation \( \Rightarrow \) a partition into equivalence classes.
  - Strongly connected components: sets of nodes which are strongly connected.

Example.

- Collapsing strongly connected components (SCCs)...
  - yields a DAG!

Finding a source.

- Property: The node with the highest post order number is in a source component.

Property++: If \( C \) and \( C' \) are SCCs with an edge from \( C \) to \( C' \),
- highest post# of a node in \( C \) larger than post# of any node in \( C' \).

Dag of SCCs

- Property: Every directed graph is a DAG of strongly connected components.

Finding the strongly connected components?

- Property: \( \text{explore}(u) \) visits all nodes reachable from \( u \).

Algorithm:
1. Run \( \text{explore} \) on node in sink component.
2. Output visited nodes.
3. Repeat.

- How do we find a node in the sink component?
SCC Algorithm.

**Property:** The highest post numbered node is in source component.

**Algorithm:**
1. Run explore on node in sink component.
2. Output visited nodes.
3. Repeat.

Uh... oh.

How should we fix this?

**Post from Reverse graph**

**Property**: If C and C' are SCCs with an edge from C to C', highest post# of a node in C larger than post# of any node in C'.

Does every node in C have a higher post order number than every node in C'?  
(A) Yes! (B) Not Necessarily. ... (B)

Explore(a) \(\Rightarrow\) Explore(b) \(\Rightarrow\) Explore(c) \(\Rightarrow\) Return from (c) \(\Rightarrow\) Explore(d) \(\Rightarrow\) Return from (d) ...

(c) has lower post order number than (d).

**Proof:**

If a node \(v\) in \(C\) is explored first.  
\(\text{explore}(v)\) gets to all of \(C\)  
and none of \(C'\).  
So every node in \(C\) explored after every node in \(C'\).  
= every post # in \(C\) larger than every post # in \(C'\).

If a node \(u\) in \(C\) is explored first  
All of \(C\) and \(C'\) will be explored before returning  
from \(\text{explore}(u)\).  
So \(u\) has higher post number than any node in \(C'\).  

Implies highest post numbered node is in source component. \(\blacksquare\)

Test your understanding..

**Property**: If C and C' are SCCs with an edge from C to C', highest post# of a node in C larger than post# of any node in C'.

**Post numbers of SCCs**

**Property++**: If C and C' are SCCs with an edge from C to C', highest post# of a node in C larger than post# of any node in C'.

Highest post# in C bigger than any in C'  
not true every post# in C greater than any in C'

Back to SCC Algorithm.

**Property**: The highest post numbered node is in source component.

**Algorithm:**
1. Run explore on node in sink component.
2. Output visited nodes.
3. Repeat.

Uh... oh.

How should we fix this?
All sinks from one dfs.

Property++: If \( C \) and \( C' \) are SCCs with an edge from \( C \) to \( C' \), highest post# of a node in \( C \) larger than post# of any node in \( C' \).

First explore of \( G \):
- Removes sink component of \( G \).
- \( \Rightarrow \) removes source component of \( G^R \).
- \( \Rightarrow \) highest rem. post # vertex, \( v \).
  - in \( G^R \) in component with no in-edges
  - \( \Rightarrow \) in source component of \( G^R \)
  - \( \Rightarrow \) \( v \) in sink component of \( G \)

SCC Algorithm:
1. DFS of \( G^R \).
2. Run undirected components algorithm on \( G \) — in reverse post order number from step 1.

\( O(|V| + |E|) \) time ...

Compute \( G^R \) in linear time?.. exercise.

Example Again: think runtime.

Property++: If \( C \) and \( C' \) are SCCs with an edge from \( C \) to \( C' \), highest post# of a node in \( C \) larger than post# of any node in \( C' \).

Lecture in a minute!

Quick Review:
- DFS so far ≡ how I learned to love the stack.
- \( \text{pre/post} = \text{time on stack} \).
- Topological Ordering:
  - Inverse post ordering = topological ordering.
  - Remove source, repeat.
- Strongly Connected Components: directed graphs.
  - Strong Connectivity for \( u \) and \( v \).
  - On a cycle together.
  - Easy: \( O(|V| + |E|) \) algorithm.
  - Linear time algorithm!
  - Observation: Highest post in "source component".
  - Find vertex in sink component.
  - Explore.
  - Repeat.