Lecture: Analysis of Algorithms (CS583 - 002)[1]

Amarda Shehu

Fall 2017

[1] Some material adapted from Kevin Wayne's Algorithm Class @ Princeton

1. Classic Applications of DSF
   - Topological Sort
   - Strongly Connected Components

Graph-Search Problem: Topological Sort

The pirates of a ship in the Mediterranean just looted a small island off the Adriatic coast. They got away with a bag of N items.

Realizing that some items are more precious than others, they are trying to come up with a fair scheme to distribute them.

Surprisingly, these pirates are fair and sensitive to bullying. They think the rule should be simple: if pirate A has ever bullied pirate B, B gets to choose before A.

Can you help them out?

Graph-Search Problem: Topological Sort

Problem Statement: Given a directed acyclic graph (DAG) $G = (V, E)$, obtain a linear ordering of V such that if $(u, v) \in E$, then u appears before v in the ordering.

Applications: First in context of PERT technique for scheduling in project management (Jarnagin 1960). Typical applications: instruction scheduling, order of compilation tasks in makefiles, resolving symbol dependencies in linkers, and task scheduling.

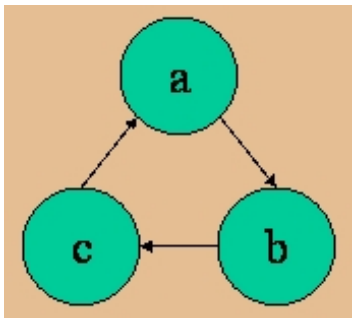


Figure: which can be scheduled first?

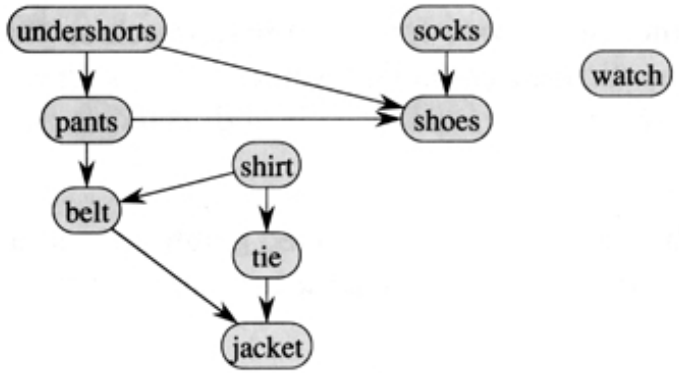


Figure: what to wear first?

A Solution to Topological Sort

Observations:

- Consider the in-degree of vertices
- The first vertex in a topological sort must have in-degree 0
- There is such a vertex in every DAG

A Solution:

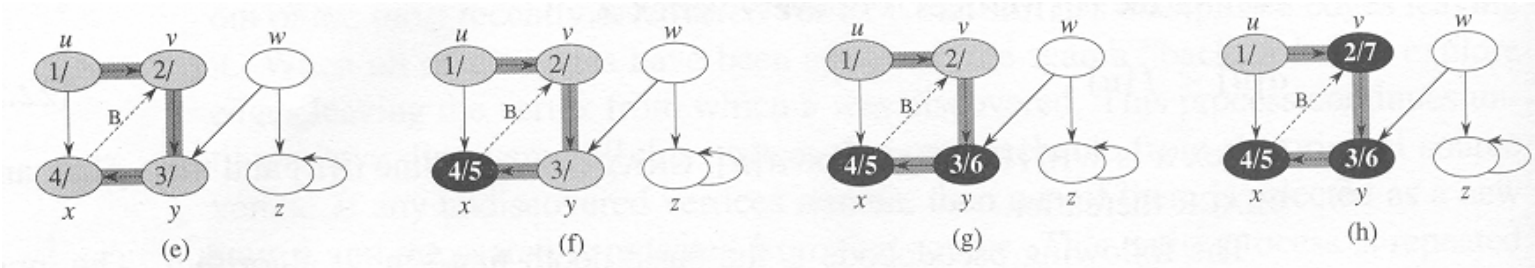
- Repeat the following steps until the graph is empty:
  1. Select a vertex v that has in-degree 0
  2. Print or store v in some list at index i, then increment i
  3. Remove v and all edges emanating from v from graph

Questions:

- Why is this solution correct?
- What is the running time of this solution?

A DFS-based Solution to Topological Sort

- A topological sort is a linear sort on finishing times
- The finishing time of a vertex records when DFS has found everything reachable from it
- If there is an edge (u, v) in a DAG, then $f[u] >= f[v]$
- The ordering in topological sort requires that vertices with higher finishing times be printed first



Strategy:

1. As a vertex is finished, place it to front of a linked list

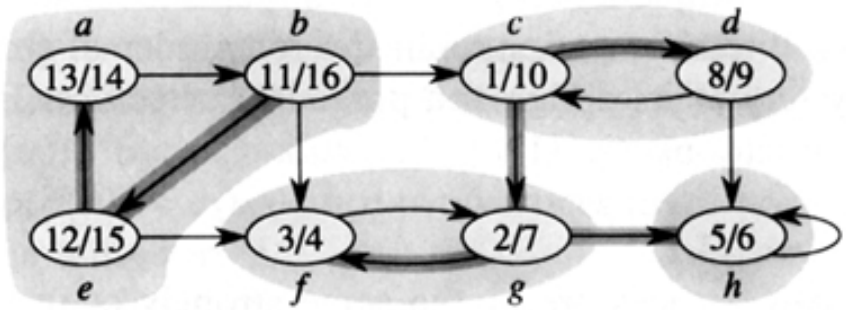
Questions: Correctness, Running Time

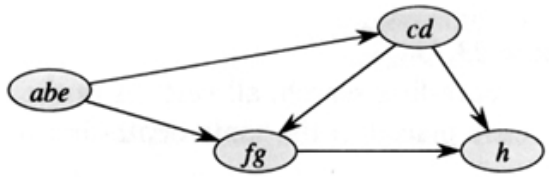
Graph-Search Problem: Strongly Connected Components

A strongly connected component (SCC) of a directed graph $G = (V, E)$ is a maximal set of vertices $C \subseteq V$ such that for every pair of vertices u, v in C, both u and v are reachable from one another.

Problem Statement: Find SCCs of G

Applications: Computer vision, image segmentation, labeling, decomposing social networks

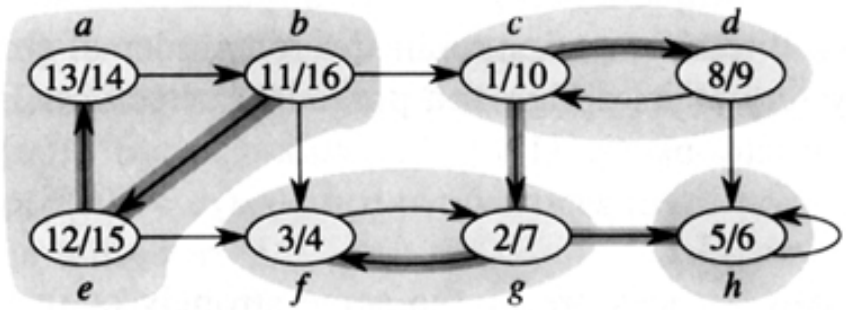


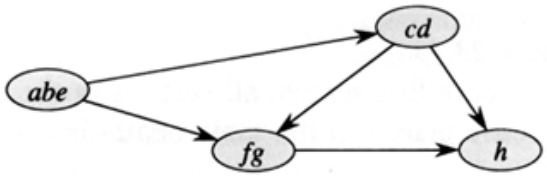


Key observation: If each SCC is contracted to a single vertex, the resulting graph is a dag.

Important Observations

1. One DFS run on a graph will reveal one connected component
2. DFS started at a node u will get stuck and needs to be restarted when all nodes reachable from u have been visited
3. Define a sink SCC as an SCC with no edges leaving it. Find one such SCC in the graph below. What does it correspond to in the dag on the right? What about the tranpose of the dag?
4. What happens if DFS is started from a node in a sink SCC?
5. Does this give you an idea for how to find SCCs?



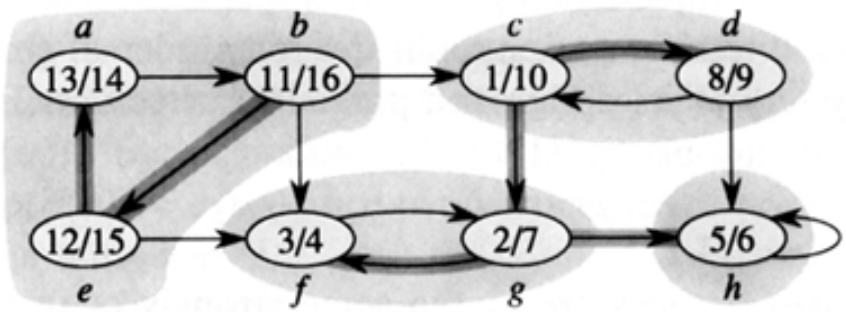


Important Observations

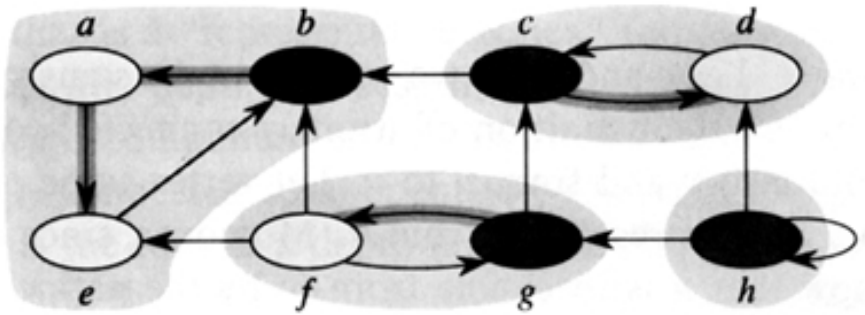
1. Identify a node in a sink SCC. Run DFS on that node and output the nodes visited. They will be all the nodes in that sink SCC and nothing else. Why?
2. Remove the sink SCC and all connections involving it.
3. You will now have a new sink SCC (why?). Go to step 1.

An important question remains: How to identify a node in a sink SCC? **Note:** G^T and G have exactly the same SCCs

Given graph $G = (V, E)$



Transpose graph $G^T = (V, E^T)$

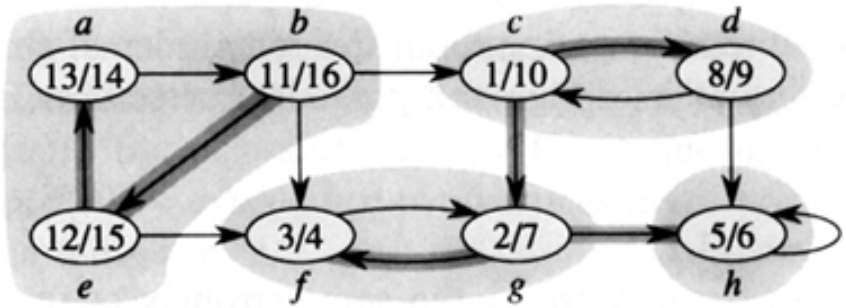


How to Identify a Node in a Sink SCC?

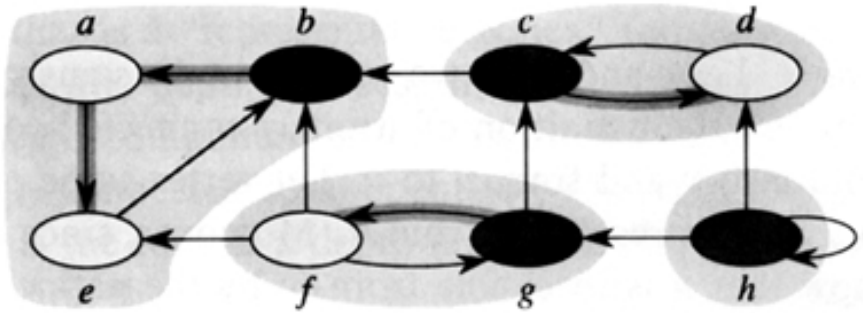
- G^T and G have exactly the same SCCs
- A sink SCC in G is a source SCC in G^T (a source SCC has no edges coming to it)
- Node with highest DFS finish time in G^T belongs to a source SCC in G^T (which is a sink SCC in G)
- This node is a sink node SCC in G on which we can start our algorithm
- Moral: Run DFS on G^T to obtain the node with the highest finish time. Run DFS on G starting from this node to get its SCC. Remove this SCC. Run DFS on modified G from highest finish time node among remaining nodes.

Putting it All Together: Finding SCCs

Given graph $G = (V, E)$



Transpose graph $G^T = (V, E^T)$
(arrows reversed)



Summary:

Step 0: Construct G^T from G

Step 1: DFS on G^T can give finishing times $f[u]$ for $u \in V$

Step 2: Run DFS on G starting from vertices with highest $f[u]$ value

Step 3: Remove SCC outputted by the DFS in step 2

Step 4: Go to step 2 on modified G

Running time is similar to DFS, adding the time it takes to construct G^T. Time to create G^T from G is $O(|V| + |E|)$. Why?