Strongly Connected Components

An instructional graph algorithm

Graham Poulter

gpoulter@cs.uct.ac.za

Department of Mathematics and Applied Mathematics

University of Cape Town

4 August 2005

The Problem

- A strongly connected component is a maximal subgraph in which there is a path from each vertex to any other vertex.
- An algorithm to find the strongly connected components of a directed graph G having n vertices and m edges in $O(n + m)$ time.
- Images by Rashid bin Muhammad.

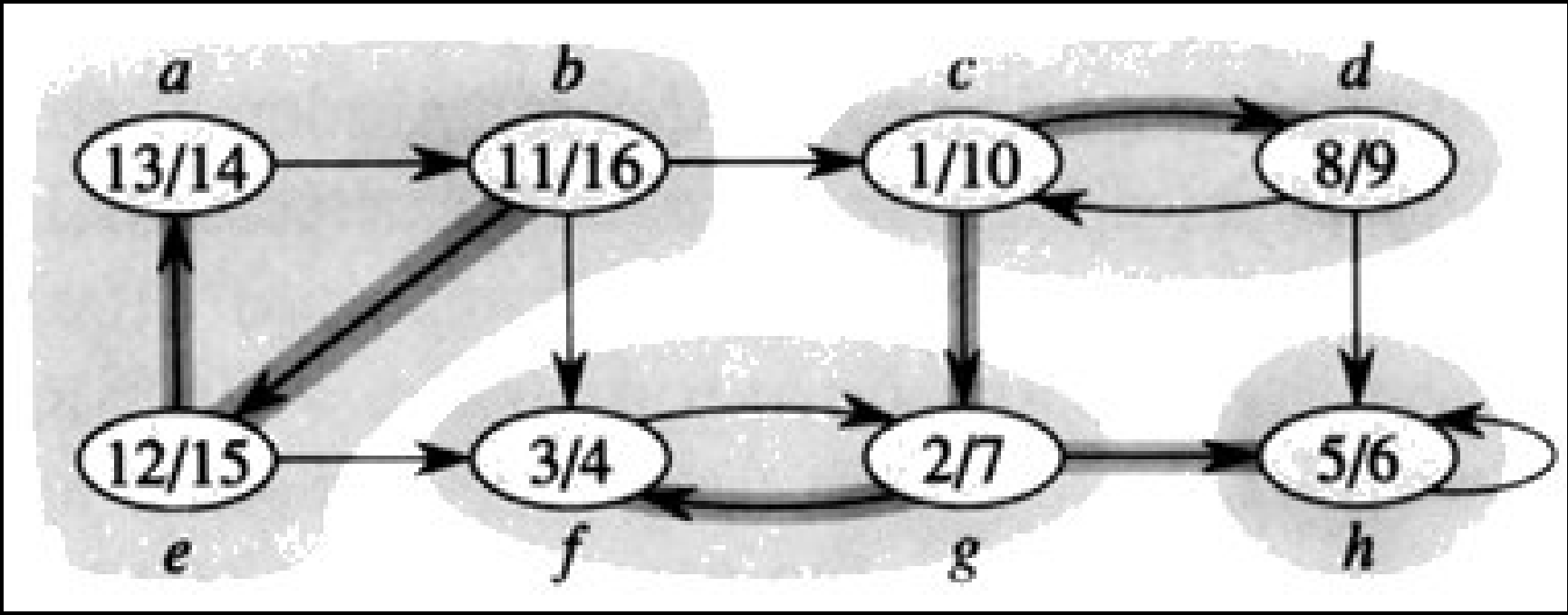
Algorithm Overview

- Use DFS to list the vertices in order of their DFS finishing times (achieved by adding the vertex to the list when its recursive DFS "visit" call returns, or simulating this with a stack).
- Transpose the directed graph to G^T by reversing the direction of each edge: every $a \rightarrow b$ becomes $b \rightarrow a$.
- Call DFS "visit" on each (unvisited) vertex in G^T, starting with the latest-finishing-time vertex from the first DFS and working backwards.

Gathering Results

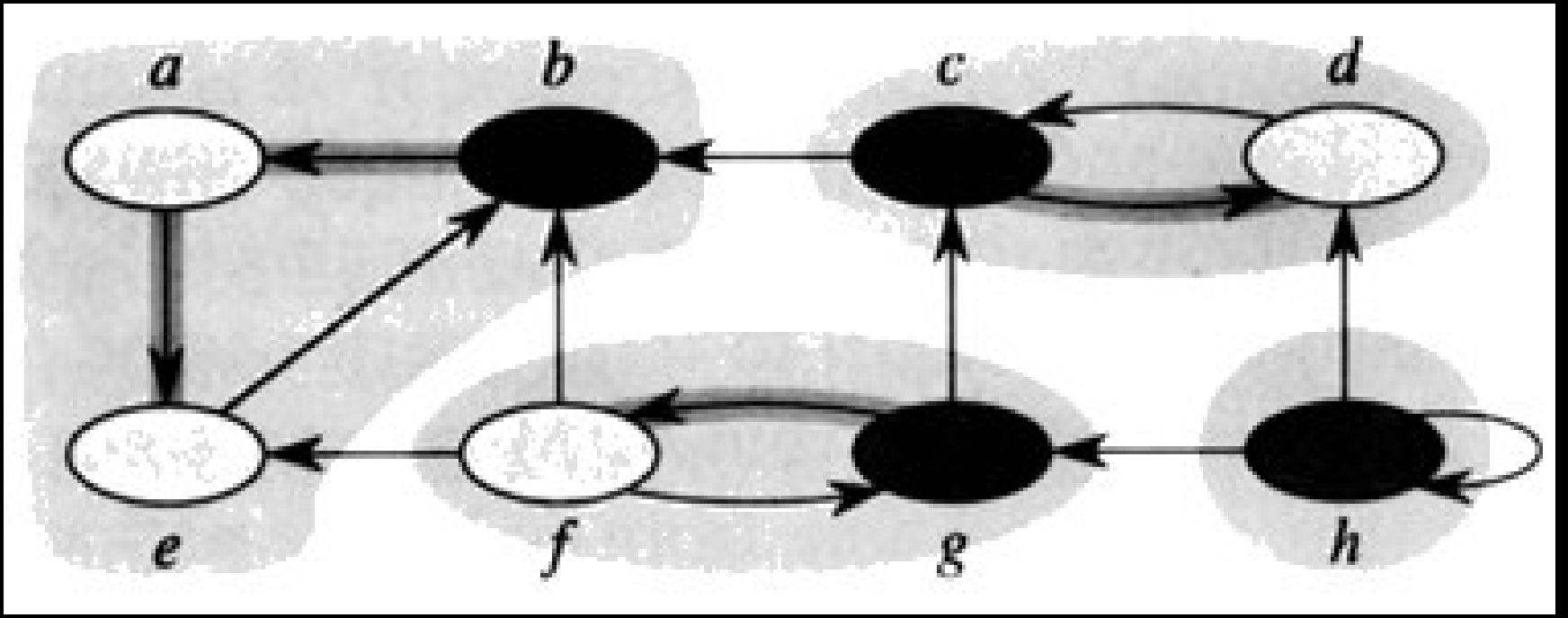
- First note that DFS on a directed graph results in a forest of DFS trees, since not all parts may be reachable from a given start vertex.
- In the second DFS, the outer loop should go backwards in the list of vertices (that is, call "visit(G^T,v)" where v is the unvisted vertex with latest finishing time). The tree will only cover vertices in a strongly connected component.

DFS Finishing Times



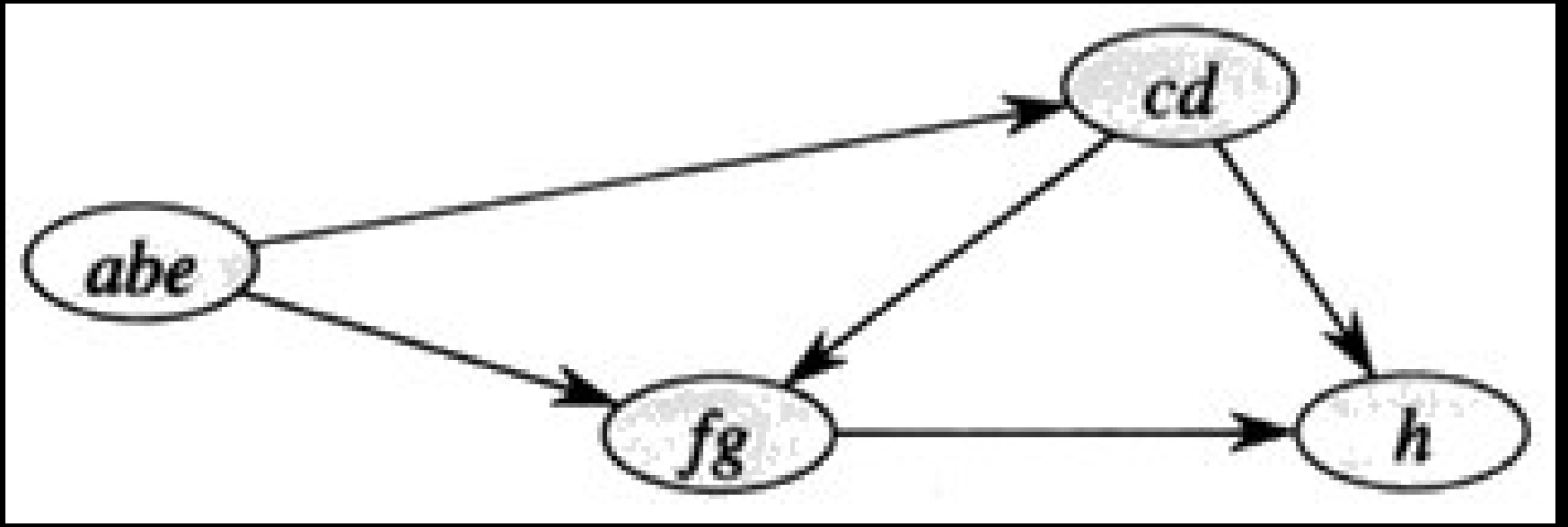
The number on the left is the start time (when "visit" was called) and the one on the right is the finish time (when "visit" returned). Strong components are grouped in clouds.

Transposed Graph



Note reversed edges. Black vertices are starting points for “visit” in the second DFS (order: b,c,g,h).

DAG of Strong Components



Each strong component can be mapped to a vertex, and the graph of components is a Directed Acyclic Graph (DAG).

Why it works

- Construct the DFS forest (with discovery, back and side edges), with later nodes and trees placed to the right of earlier ones. Hence, side edges will always point to the left.
- Mark all nodes unvisited, and reverse the edges, but leave the tree the same. The sides edges now point to the right. At each outer-loop call to visit, the “latest-finishing of the unvisited nodes” will always be the root of the rightmost tree, so there is no possibility of the second DFS escaping its strong component.

DAGness of component graph

Suppose the component graph were not acyclic. Then there would be a cycle involving two components, and the vertices involved in the cycle would be strongly connected. Such strong spanning of multiple strong components contradicts their componentness (each vertex is a member of at most one component). Hence the component graph must be acyclic.

References

- Strong connectivity

  “CPS 130: Fall 2001: Introduction to the Design and Analysis of Algorithms: Lecture 16: Graph Algorithms”
  Michael L. Littman (2001)

  www.cs.duke.edu

- Website of Rashid bin Muhammad