Biconnected 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

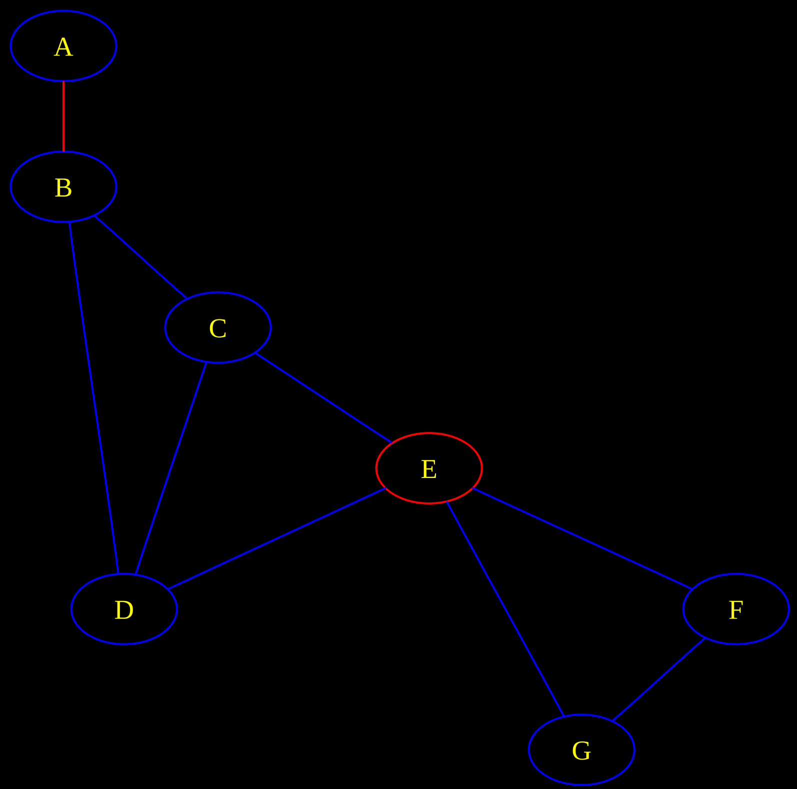
Given a connected graph G with n verticies and m edges. We shall find, in $O(n+m)$ time, the

- Biconnected components
- Separation vertices
- Separation edges

Definitions

- A separation vertex or edge is one whose removal disconnects G.
- A biconnected component is a maximal biconnected subgraph of G. Edges and non-separation vertices belong to exactly one component, while separation vertices belong to at least two.
- Biconnected components contain no separation vertices or edges (nothing to break it). Between any two vertices there exists at least two disjoint paths, and G has a simple cycle containing them.

Example Graph



Separation vertex and edge are shown in red.

Equivalence Classes

- Edges e and f of G are said to be linked if either $e = f$ (they're the same edge) or G has a simple cycle containing e and f.
- If e and f are linked, and f and g are linked, then e and g are linked (transitivity), because you can construct a cycle around them. A set of such mutually-linked edges forms an equivalence class (each edge is "equivalent" to the others in the class).
- Each equivalence class corresponds to a biconnected component of G.

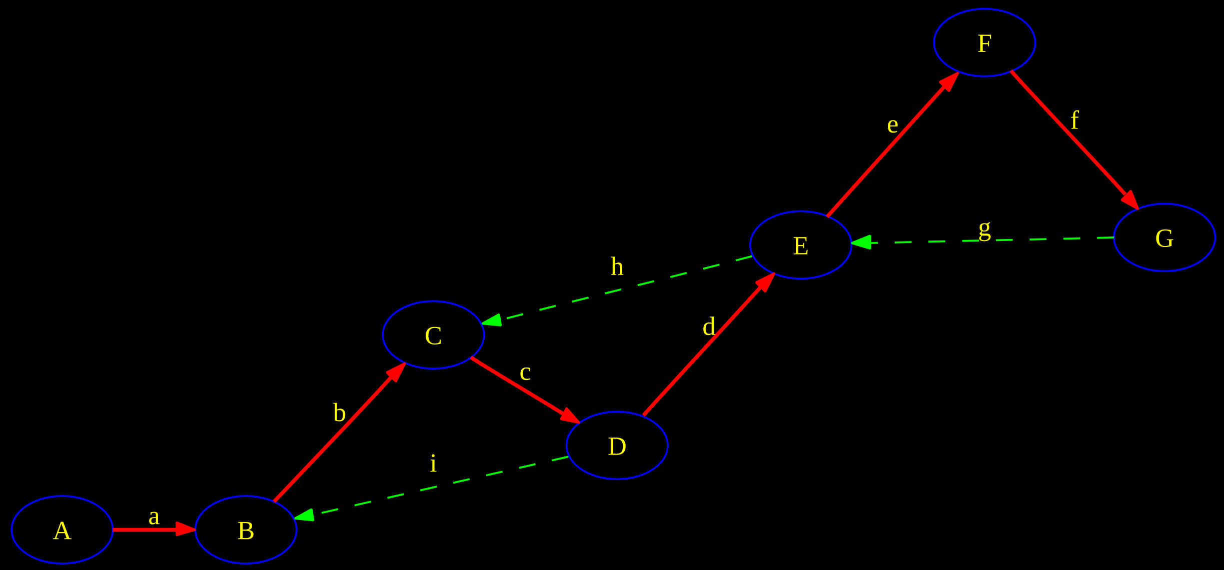
Auxiliary Graph

- Vertices in the auxiliary graph F are edges in G. We link vertices in F according to the link relation: e and f are linked if G has a simple cycle containing them.
- Each component of F represents an equivalence class, which tells us the edges in the corresponding biconnected component of G.
- Isolated vertices of F are separation edges in G. A separation vertex in G has adjacent edges whose vertices in F are in different different components.

Algorithm Overview

- Do a Depth-First Search (DFS) on G, using it to construct a *proxy graph*, F', that contains just enough links to have the same components as F.
- On the next slide (the DFS tree for G), back-edges are in dashed green, discovery edges in bold red.
- Three slides from now (the proxy graph F'), green vertices represent back-edges, and red represents discovery edges.

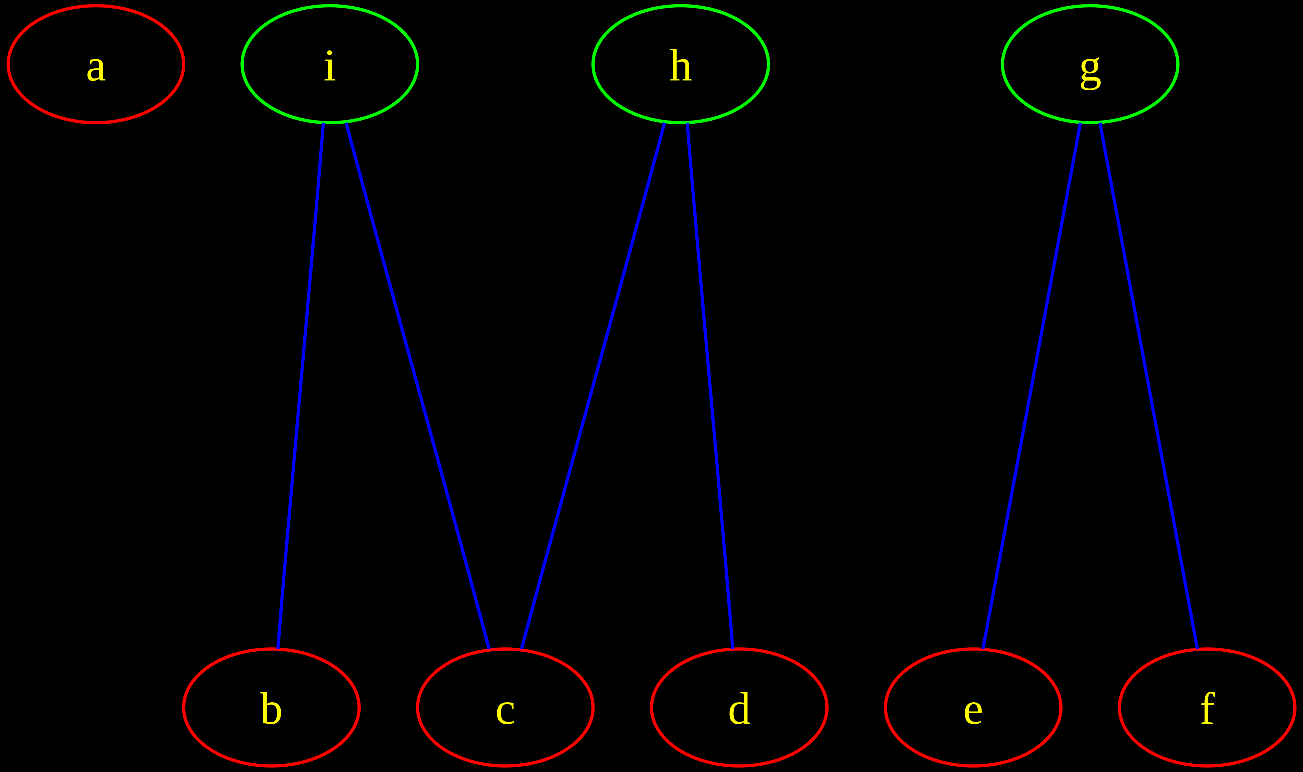
DFS Representation



Constructing the Proxy Graph

- Visit vertices v in DFS order. For each back-edge $(u \to v)$, link $(u \to v)$ to the (unique) discovery edge $(x \to u)$. Traverse backwards, linking $(u \to v)$ to ancestral discovery edges, until encountering the “root” vertex v.
- **BUT:** also stop after linking to a discovery edge that has already been linked to. There is no need to carry on once you’ve joined up with the rest of the equivalence class.

Proxy Graph F'



Proxy Graph Algorithm

for Vertices v of G in DFS order (start vertex s **do**

  for all Back-edges $e \leftarrow (u, v)$ **do**

    while $u \neq v$ **do**

      $f \leftarrow$ Discovery edge (x, u)

      $F'.\mathrm{addEdge}(e, f)$

      if $f.\mathrm{linked} = \mathrm{false}$ **then**

        $f.\mathrm{linked} \leftarrow \mathrm{true}$

        $u \leftarrow x$

      else

        $u \leftarrow v$

      end if

    end while

References

- “Algorithm Design: Foundations, Analysis, and Internet Examples”
  Michael T. Goodrich and Roberto Tamassia
  John Wiley & Sons (2002)

  http://ww3.algorithmdesign.net/handouts/Biconnectivity.pdf