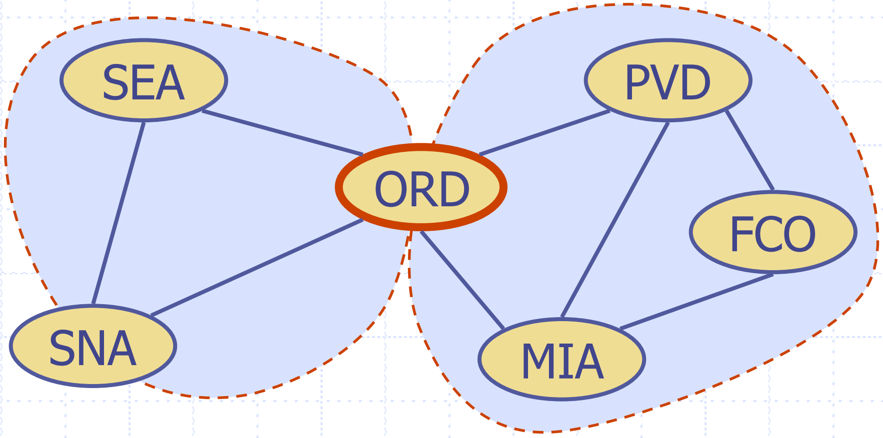
Presentation for use with the textbook, Algorithm Design and Applications, by M. T. Goodrich and R. Tamassia, Wiley, 2015

Biconnected Components

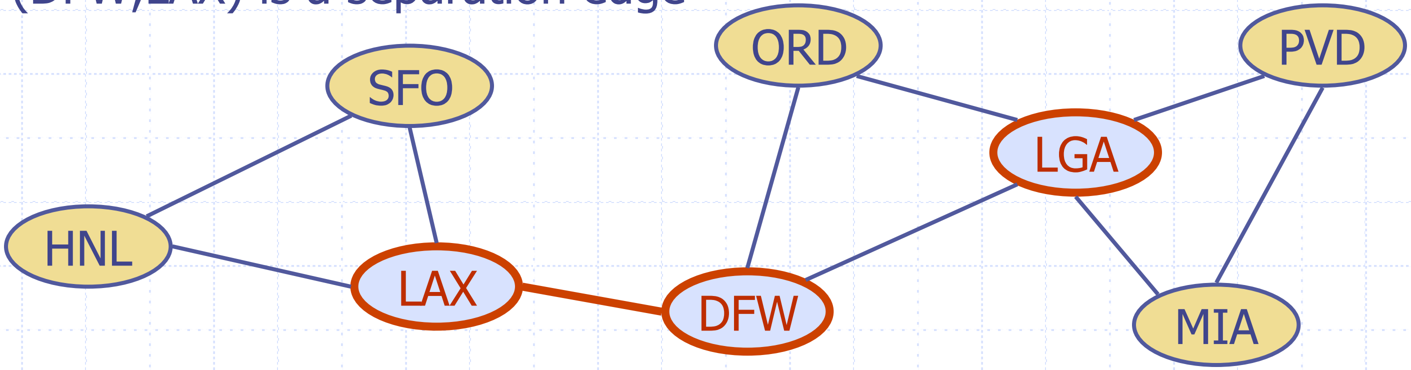


Application: Networking

- A computer network can be modeled as a graph, where vertices are routers and edges are network connections between edges.
- A router can be considered **critical** if it can disconnect the network for that router to fail.
- It would be nice to identify which routers are critical.
- We can do such an identification by solving the biconnected components problem.

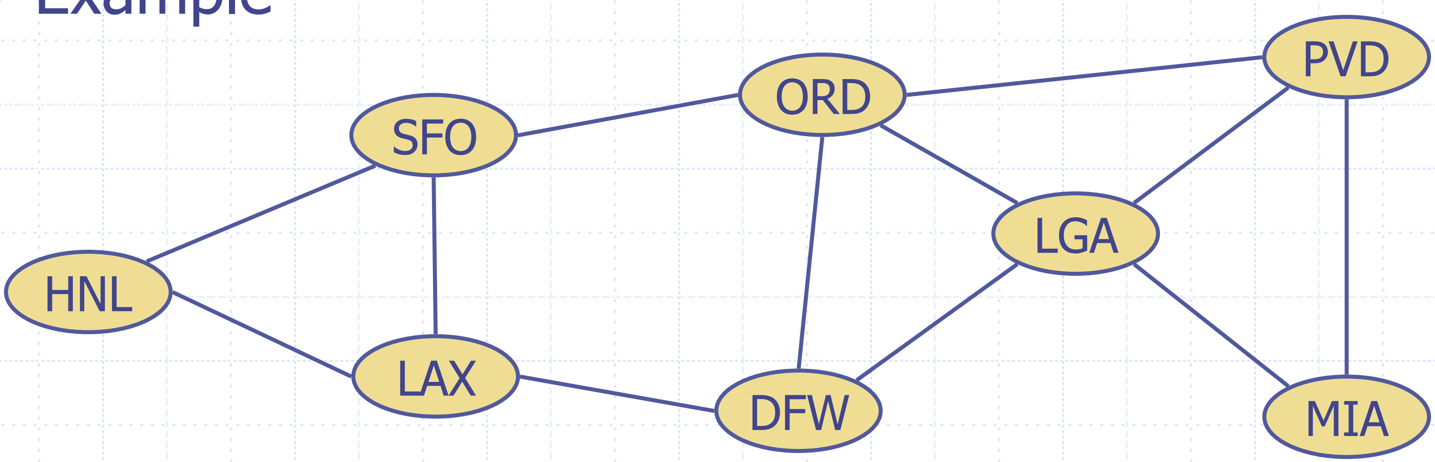
Separation Edges and Vertices

- Definitions
  - Let G be a connected graph
  - A separation edge of G is an edge whose removal disconnects G
  - A separation vertex of G is a vertex whose removal disconnects G
- Applications
  - Separation edges and vertices represent single points of failure in a network and are critical to the operation of the network
- Example
  - DFW, LGA and LAX are separation vertices
  - (DFW,LAX) is a separation edge



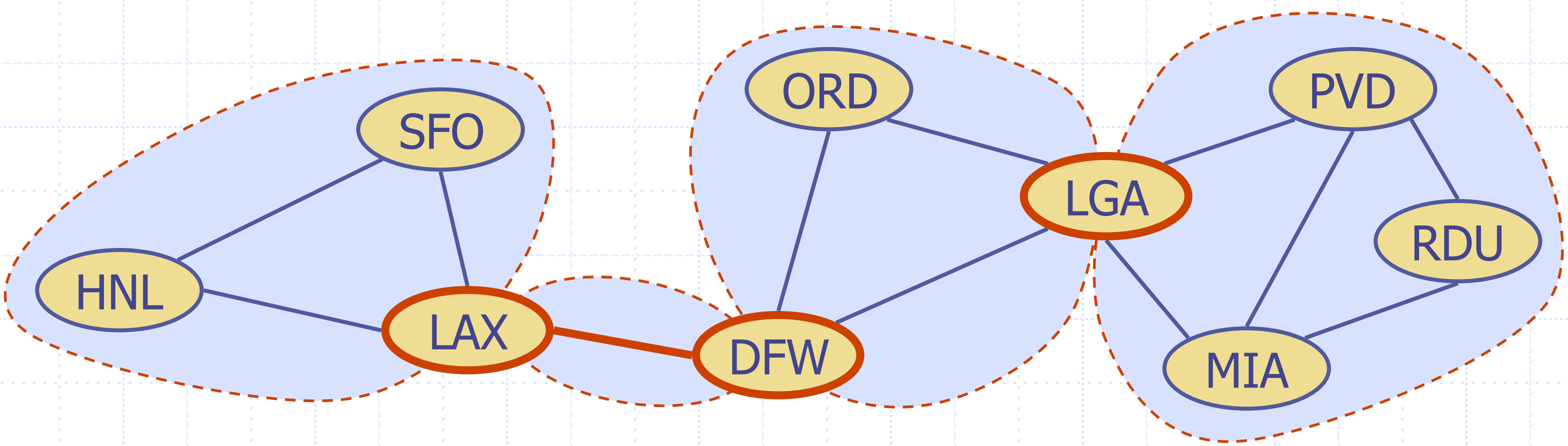
Biconnected Graph

- Equivalent definitions of a biconnected graph G
  - Graph G has no separation edges and no separation vertices
  - For any two vertices u and v of G, there are two disjoint simple paths between u and v (i.e., two simple paths between u and v that share no other vertices or edges)
  - For any two vertices u and v of G, there is a simple cycle containing u and v
- Example



Biconnected Components

- Biconnected component of a graph G
  - A maximal biconnected subgraph of G, or
  - A subgraph consisting of a separation edge of G and its end vertices
- Interaction of biconnected components
  - An edge belongs to exactly one biconnected component
  - A nonseparation vertex belongs to exactly one biconnected component
  - A separation vertex belongs to two or more biconnected components
- Example of a graph with four biconnected components



Equivalence Classes

- Given a set S, a relation R on S is a set of ordered pairs of elements of S, i.e., R is a subset of $S \times S$
- An equivalence relation R on S satisfies the following properties

  Reflexive: $(x,x) \in R$

  Symmetric: $(x,y) \in R \Rightarrow (y,x) \in R$

  Transitive: $(x,y) \in R \wedge (y,z) \in R \Rightarrow (x,z) \in R$
- An equivalence relation R on S induces a partition of the elements of S into equivalence classes
- Example (connectivity relation among the vertices of a graph):
  - Let V be the set of vertices of a graph G
  - Define the relation
    $C = \{(v,w) \in V \times V$ such that G has a path from v to $w\}$
  - Relation C is an equivalence relation
  - The equivalence classes of relation C are the vertices in each connected component of graph G

Link Relation

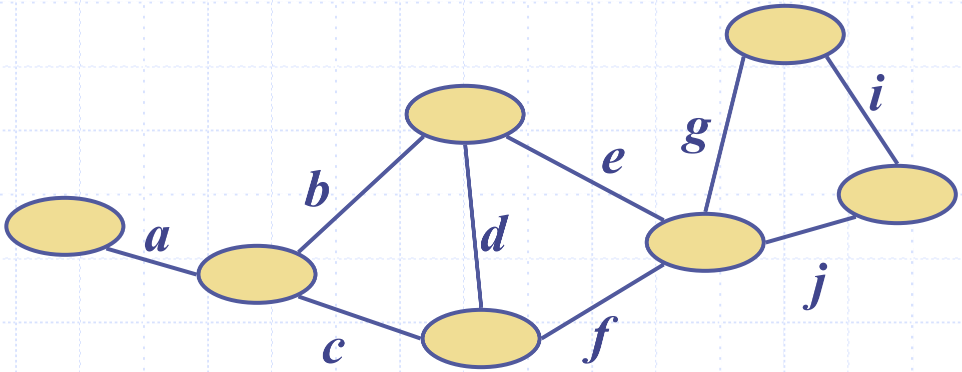
- Edges e and f of connected graph G are linked if
  - $e = f$, or
  - G has a simple cycle containing e and f

Theorem:

The link relation on the edges of a graph is an equivalence relation

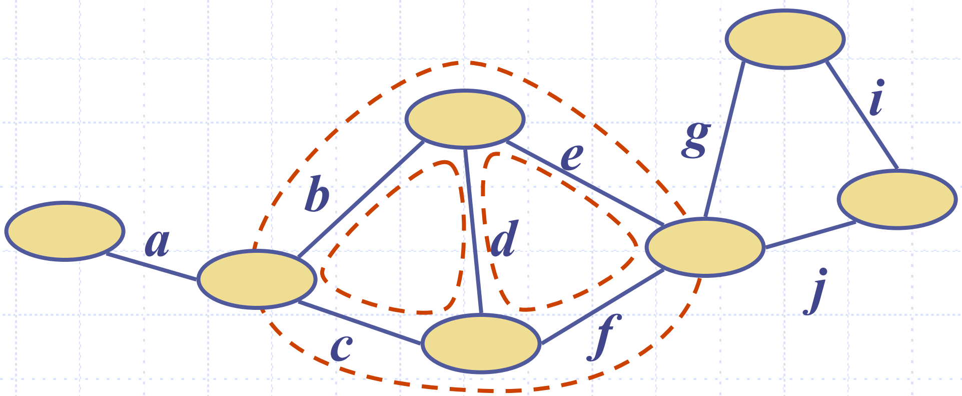
Proof Sketch:

- The reflexive and symmetric properties follow from the definition
- For the transitive property, consider two simple cycles sharing an edge



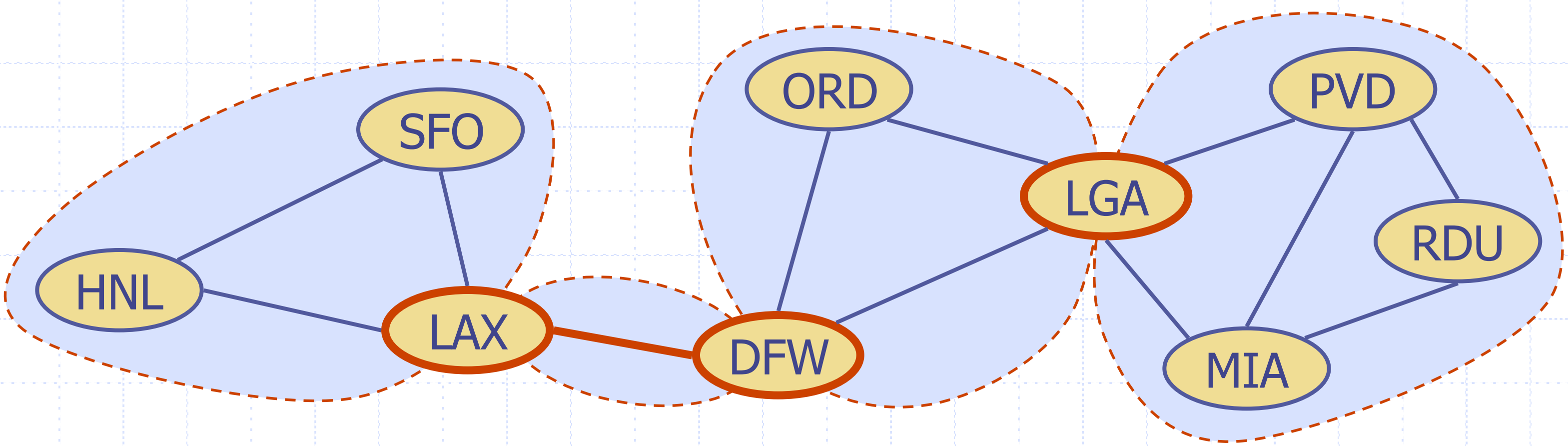
Equivalence classes of linked edges:

$\{a\}$ $\{b, c, d, e, f\}$ $\{g, i, j\}$



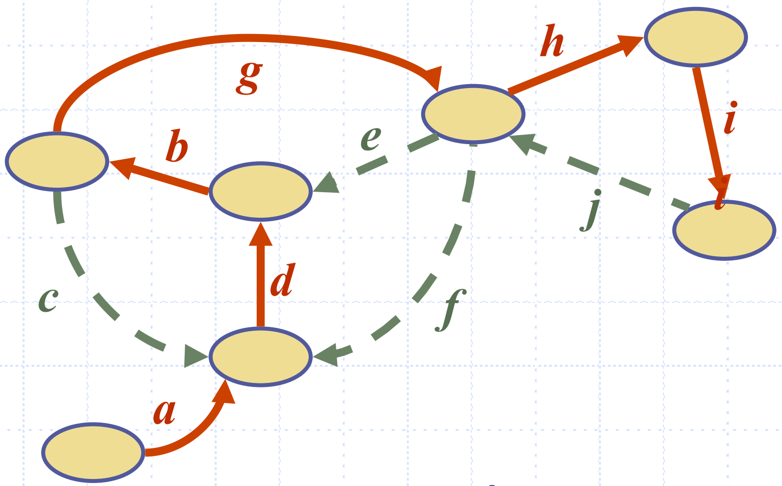
Link Components

- The link components of a connected graph G are the equivalence classes of edges with respect to the link relation
- A biconnected component of G is the subgraph of G induced by an equivalence class of linked edges
- A separation edge is a single-element equivalence class of linked edges
- A separation vertex has incident edges in at least two distinct equivalence classes of linked edge

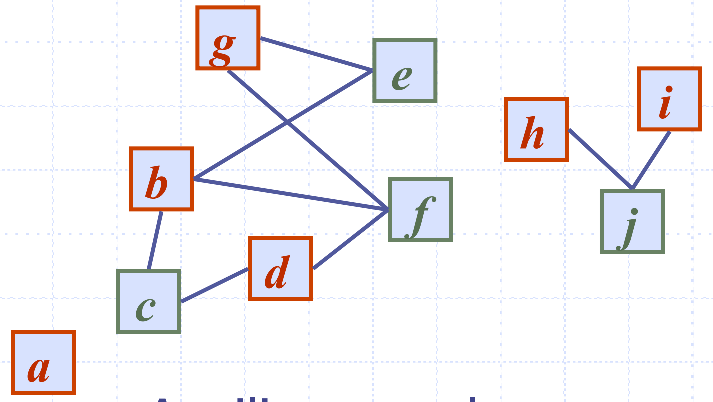


Auxiliary Graph

- Auxiliary graph B for a connected graph G
  - Associated with a DFS traversal of G
  - The vertices of B are the edges of G
  - For each back edge e of G, B has edges $(e,f_1), (e,f_2), \ldots, (e,f_k)$, where $f_1, f_2, \ldots, f_k$ are the discovery edges of G that form a simple cycle with e
  - Its connected components correspond to the the link components of G



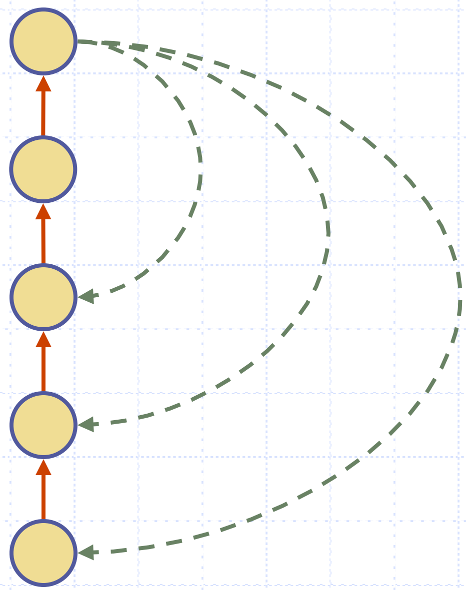
DFS on graph G



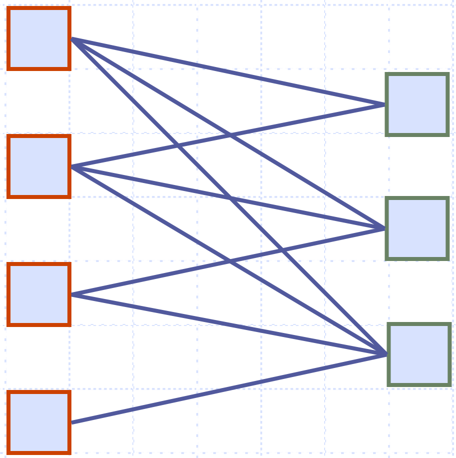
Auxiliary graph B

Auxiliary Graph (cont.)

- In the worst case, the number of edges of the auxiliary graph is proportional to nm



DFS on graph G



Auxiliary graph B

An O(nm)-Time Algorithm

- Lemma: The connected components of the auxiliary graph B correspond to the link components of the graph G that induced B.
- This lemma yields the following O(nm)-time algorithm for computing all the link components of a graph G with n vertices and m edges:

1. Perform a DFS traversal T on G.
2. Compute the auxiliary graph B by identifying the cycles of G induced by each back edge with respect to T.
3. Compute the connected components of B, for example, by performing a DFS traversal of the auxiliary graph B.
4. For each connected component of B, output the vertices of B (which are edges of G) as a link component of G.

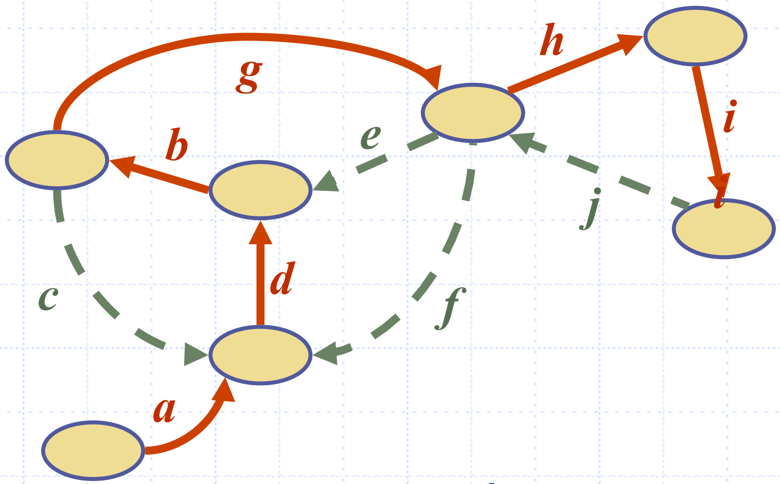
A Linear-Time Algorithm

Algorithm LinkComponents(G):
    Input: A connected graph G
    Output: The link components of G

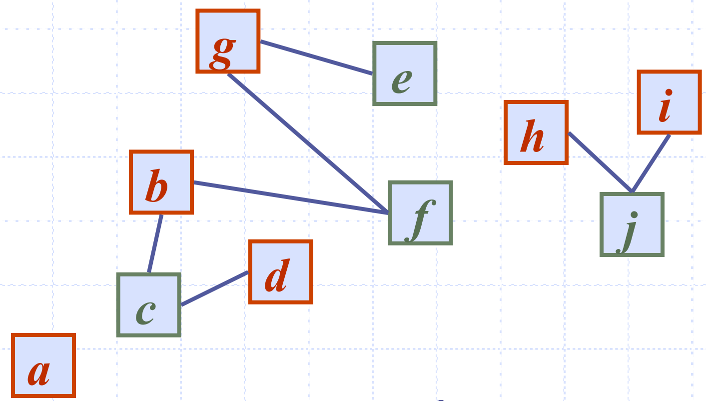
Let F be an initially empty auxiliary graph.
Perform a DFS traversal of G starting at an arbitrary vertex s.
Add each DFS discovery edge f as a vertex in F and mark f "unlinked."
For each vertex v of G, let $p(v)$ be the parent of v in the DFS spanning tree.
for each vertex v, in increasing rank order as visited in the DFS traversal **do**
    for each back edge $e = (u, v)$ with destination v **do**
        Add e as a vertex of the graph F.
        // March up from u to s adding edges to F only as necessary.
        while $u \neq v$ **do**
            Let f be the vertex in F corresponding to the discovery edge $(u, p(u))$.
            Add the edge (e, f) to F.
            if f is marked "unlinked" **then**
                Mark f as "linked."
                $u \leftarrow p(u)$
            else
                $u \leftarrow v$ // shortcut to the end of the while loop
Compute the connected components of the graph F.

Analysis with the Proxy Graph, F

- Proxy graph F for a connected graph G
  - Spanning forest of the auxiliary graph B
  - Has m vertices and $O(m)$ edges
  - Can be constructed in $O(n + m)$ time
  - Its connected components (trees) correspond to the the link components of G
- Given a graph G with n vertices and m edges, we can compute the following in $O(n + m)$ time:
  - The biconnected components of G
  - The separation vertices of G
  - The separation edges of G



DFS on graph G



Proxy graph F