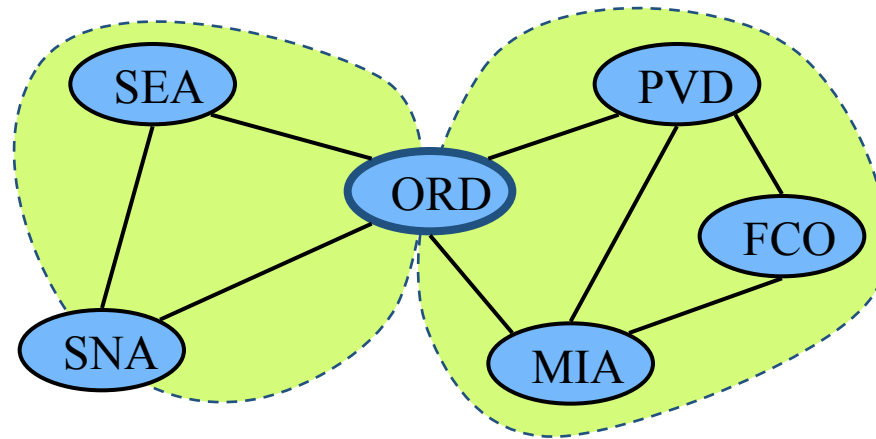


Biconnectivity



Outline and Reading

Definitions (6.3.2)

- Separation vertices and edges
- Biconnected graph
- Biconnected components
- Equivalence classes
- Linked edges and link components

Algorithms (6.3.2)

- Auxiliary graph
- Proxy graph

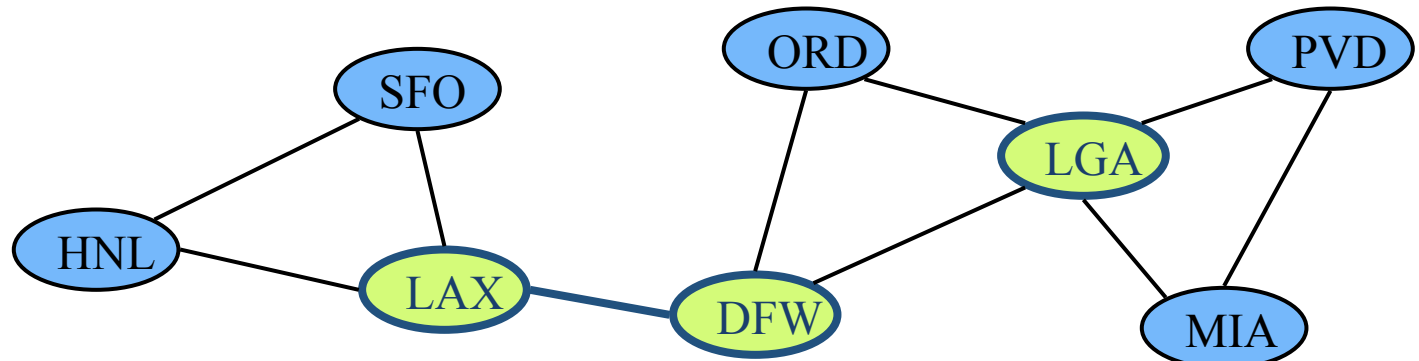
Separation Edges and Vertices

Let G be a connected graph

- A **separation edge** of G is an edge whose removal disconnects G .
Ex: (DFW,LAX) is a separation edge
- A **separation vertex** of G is a vertex whose removal disconnects G .
Ex: DFW, LGA and LAX are separation vertices

Applications:

- Separation edges and vertices **represent single points of failure** in a network and are critical to the operation of the network.

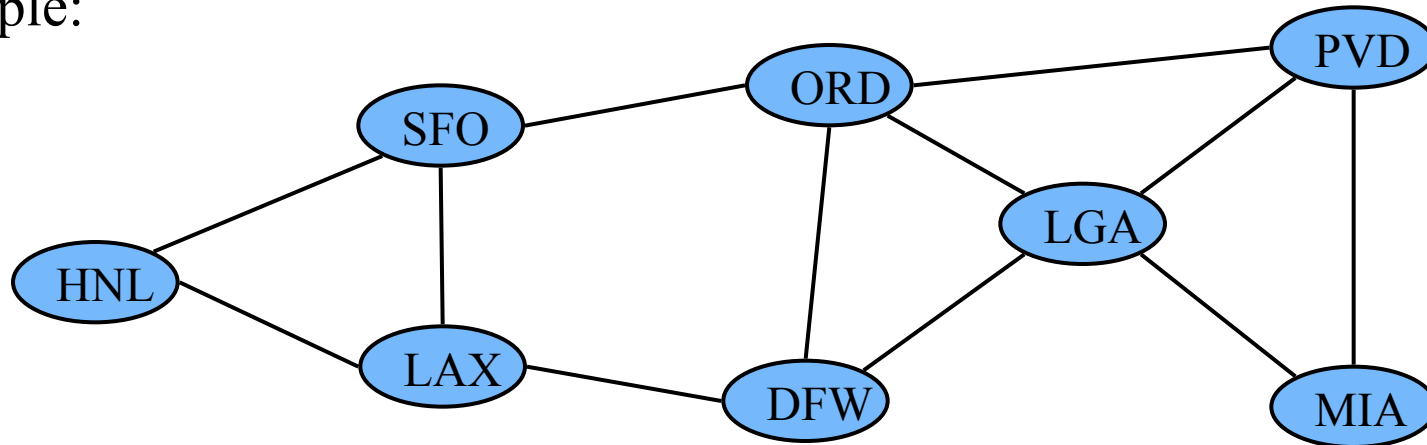


Biconnected Graph

Equivalent definitions of a **biconnected graph** G :

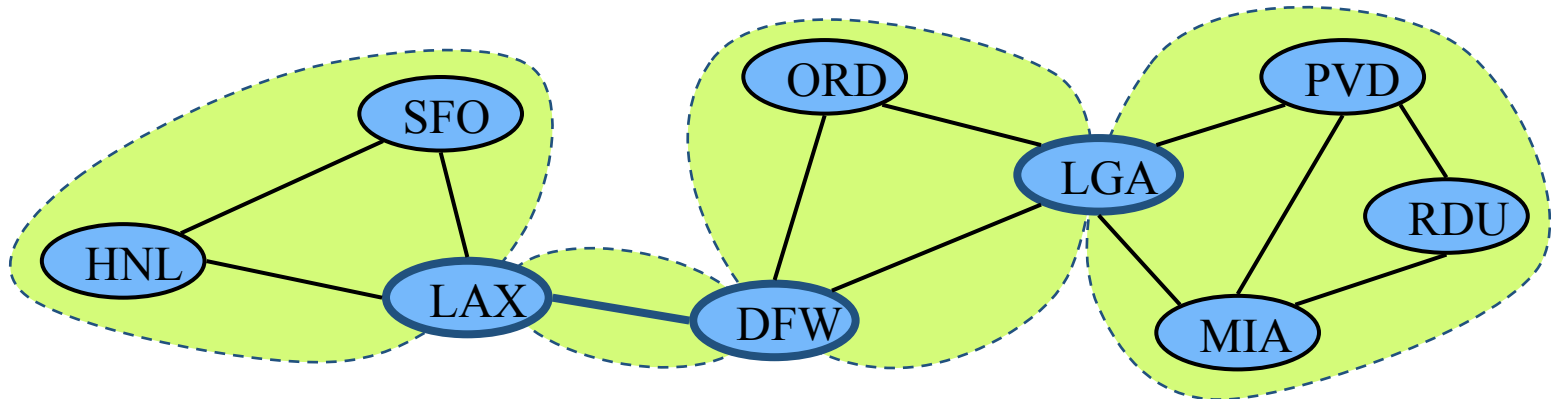
- Graph G has no separation edges and no separation vertices.
- For any two vertices u and v of G , there are two disjoint simple paths between u and v (i.e., two simple paths between u and v that share no other vertices or edges).
- For any two vertices u and v of G , there is a simple cycle containing 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:** $R(x,x)$ is true for each x
 - Symmetric:** $R(x,y) = R(y,x)$ for each x,y
 - Transitive:** $R(x,y) \wedge R(y,z) \rightarrow R(x,z)$ for each x,y,z
- 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
 - $C = \{(v,w) \in V \times V \text{ such that } G \text{ has a path from } v \text{ to } w\}$
- Relation C is an equivalence relation
- The equivalence classes of relation C are the vertices in each connected component of graph G

Link Relation

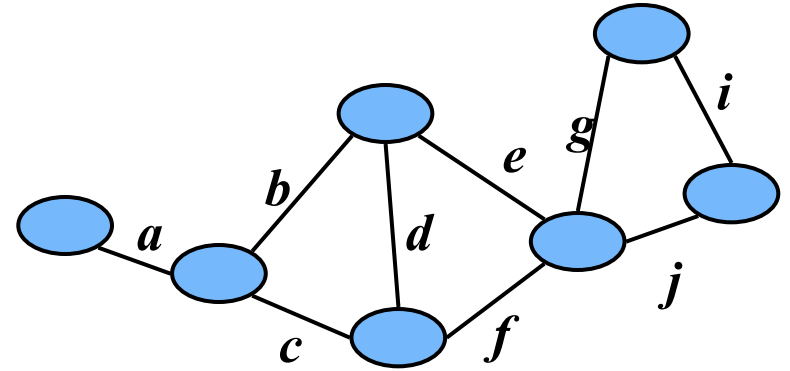
Edges e and f of connected graph G are **linked** if

- $e = f$, or
- G has a simple cycle containing e and f

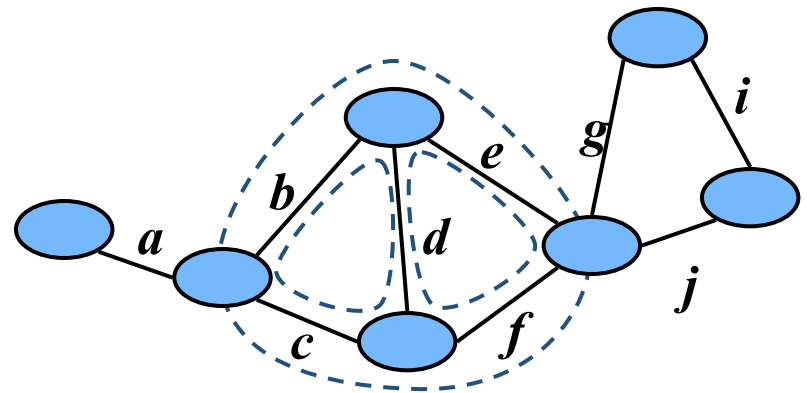
Theorem: The link relation on the edges of a graph is an equivalence relation.

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

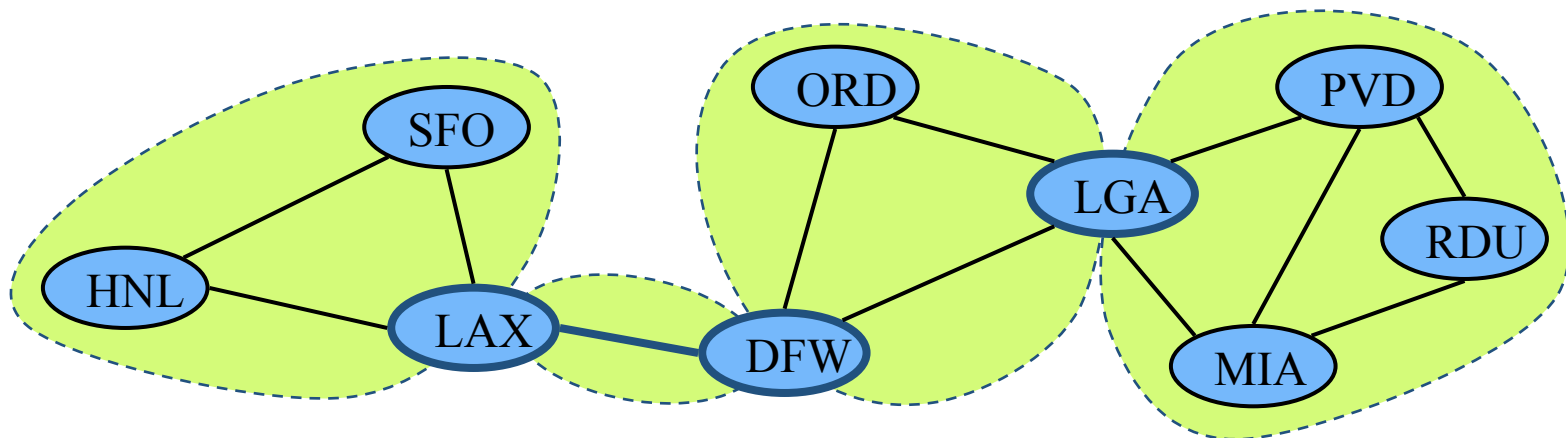


Link Components

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

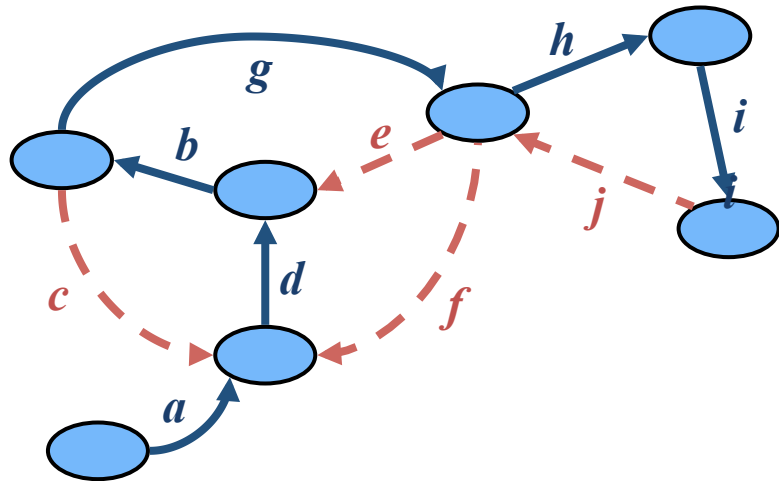


Auxiliary Graph

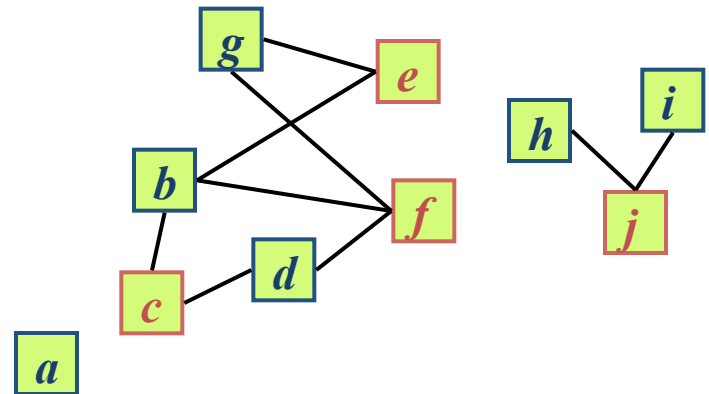
Auxiliary graph B for a connected graph G

- 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 $(e, f_1), (e, f_2), \dots, (e, f_k)$, where f_1, f_2, \dots, f_k are the discovery edges of G that form a simple cycle with e

The connected components of B correspond to the link components of G



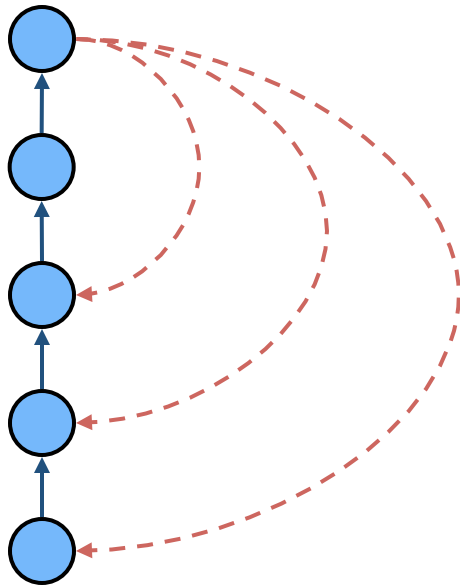
DFS on graph G



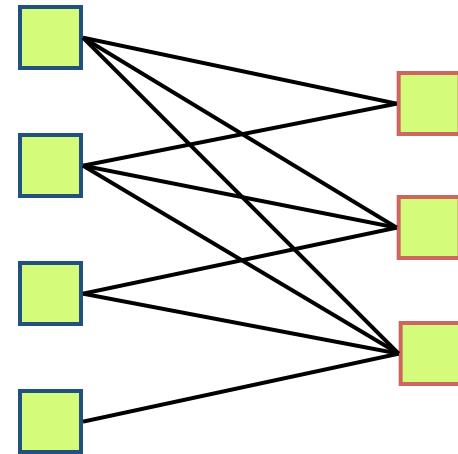
Auxiliary graph B

Auxiliary Graph (cont.)

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



DFS on graph G



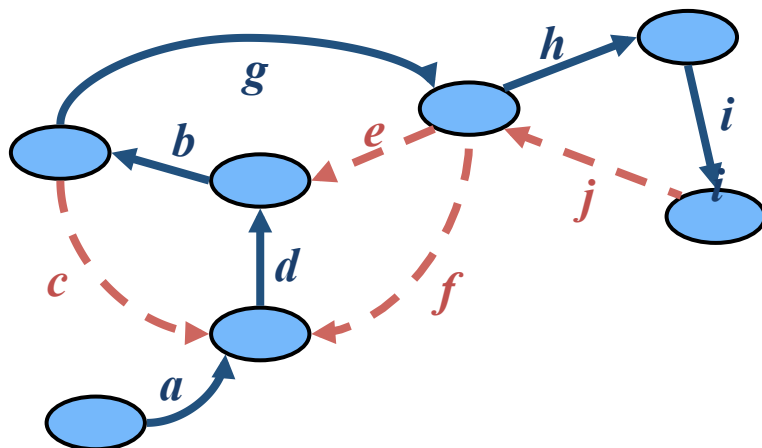
Auxiliary graph B

An Algorithm to Compute Biconnected Components

1. Perform DFS traversal on G
2. Compute auxiliary graph B
3. Compute connected components of B
4. For each connected component of B , output vertices of B (edges of G) as a link component of G

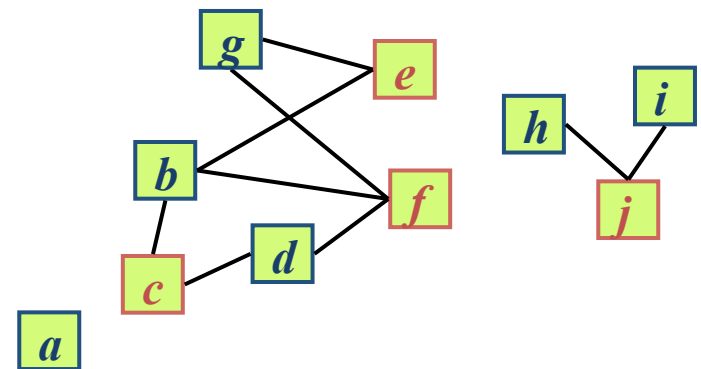
Running time is $O(nm)$. Why?

Can we do better?



DFS on graph G

Biconnectivity



Auxiliary graph B

Proxy Graph

Algorithm *proxyGraph*(G)

Input connected graph G

Output proxy graph F for G

$F \leftarrow$ empty graph

DFS(G, s) { s is any vertex of G }

for all discovery edges e of G

$F.insertVertex(e)$

setLabel($e, UNLINKED$)

for all vertices v of G in DFS visit order

for all back edges $e = (u, v)$

$F.insertVertex(e)$

repeat {add edges to F only as necessary}

$f \leftarrow$ discovery edge with dest. u

$F.insertEdge(e, f, \emptyset)$

if $F.getLabel(f) = UNLINKED$

setLabel($f, LINKED$)

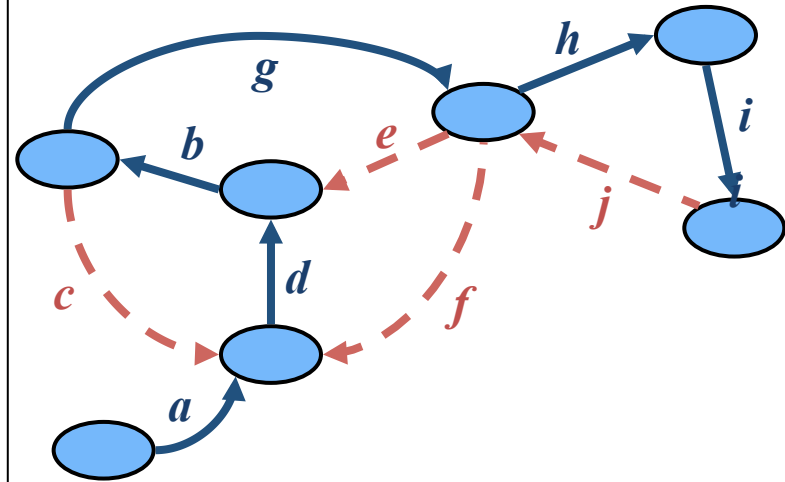
$u \leftarrow$ origin of edge f

else

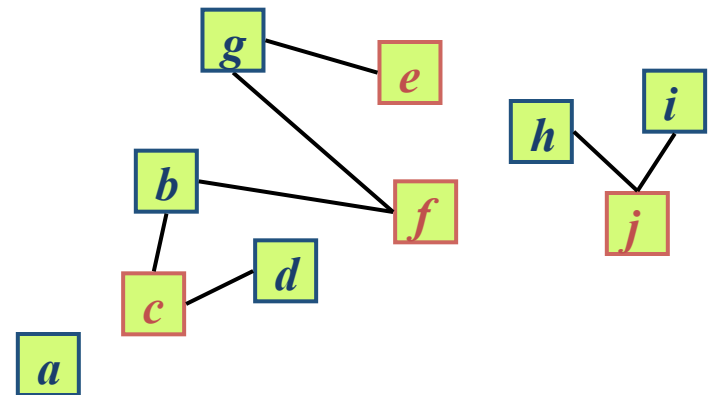
$u \leftarrow v$ { ends the loop }

until $u = v$

return F



DFS on graph G

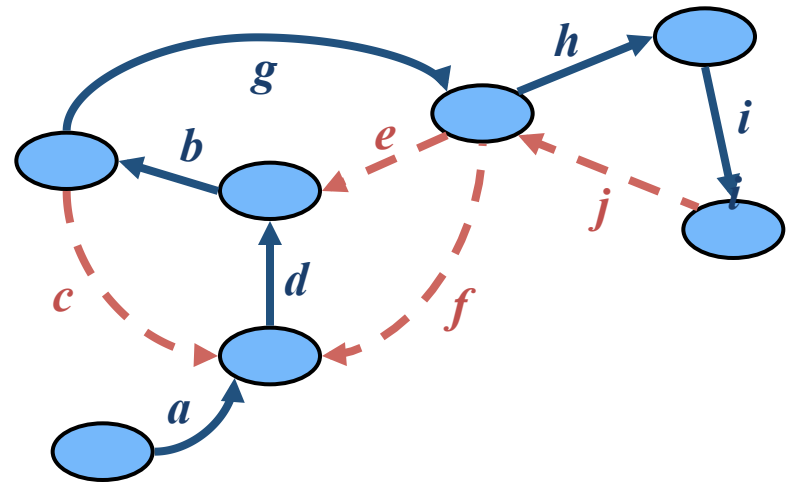


Proxy graph F

Proxy Graph (cont.)

Proxy graph F for a connected graph G

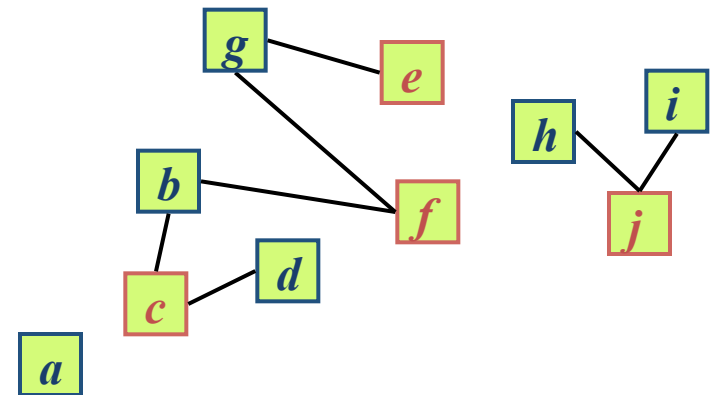
- Spanning forest of the auxiliary graph B
- Has m vertices and $O(m)$ edges
- Can be constructed in $O(n + m)$ time
- Its connected components (trees) correspond to the link components of G



DFS on graph 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



Proxy graph F