Connectivity and Biconnectivity

- connected components
- cutvertices
- biconnected components
Connected Components

**Connected Graph**: any two vertices connected by a path

**Connected Component**: maximal connected subgraph of a graph
Equivalence Relations

A relation on a set $S$ is a set $R$ of ordered pairs of elements of $S$ defined by some property.

Example:
- $S = \{1, 2, 3, 4\}$
- $R = \{(i, j) \in S \times S \text{ such that } i < j\}$
  $= \{(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)\}$

An equivalence relation is a relation with the following properties:
- $(x, x) \in R$, $\forall x \in S$ (reflexive)
- $(x, y) \in R \Rightarrow (y, x) \in R$ (symmetric)
- $(x, y), (y, z) \in R \Rightarrow (x, z) \in R$ (transitive)

The relation $C$ on the set of vertices of a graph:
- $(u, v) \in C \iff u$ and $v$ are in the same connected component

is an equivalence relation.
DFS on a Disconnected Graph

- DFS($v$) visits all the vertices and edges in the connected component of $v$

- To compute the connected components:

  $k = 0$ // component counter
  
  foreach (vertex $v$)
    if unvisited($v$)
      // add to component k
      // the vertices reached by $v$
      DFS($v$, k++)
Cutvertices

Cutvertex (separation vertex): its removal disconnects the graph

If the Chicago airport is closed, then there is no way to get from Providence to cities on the west coast.

Similarly for Denver.

• Cutvertices: ORD, DEN
Biconnectivity

Biconnected graph: has no cut vertices

New flights: LGA-ATL and DFW-LAX make the graph biconnected.
Properties of Biconnected Graphs

- There are *two disjoint paths* between any two vertices.
- There is a *cycle* through any two vertices.

By convention, two nodes connected by an edge form a biconnected graph, but this does not verify the above properties.
Biconnected Components

• Biconnected component (block): maximal biconnected subgraph

• Biconnected components are edge-disjoint but share cutvertices.
Characterization of the Biconnected Components

- **Equivalence relation** $R$ on the edges of $G$: $(e', e'') \in R$ if there is a cycle containing both $e'$ and $e''$

- Proof of the **transitive property**

- We partition the edges of $G$ into **equivalence classes** with respect to $R$.

- Each equivalence class corresponds to
  - a biconnected components of $G$
  - a connected components of a graph $H$ whose vertices are the edges of $G$ and whose edges are the **pairs** in relation $R$. 
**DFS and Biconnected Components**

- Graph H has $O(m^2)$ edges in the worst case.
- Instead of computing the entire graph H, we use a smaller *proxy* graph K.
- Start with an empty graph K whose vertices are the edges of G.
- Given a DFS on G, consider the $(m - n + 1)$ cycles of G induced by the back edges.
- For each such cycle $C = (e_0, e_1, \ldots, e_p)$ add edges $(e_0, e_1) \ldots (e_0, e_p)$ to K.

- The connected components of K are the same as those of H!
A Linear Time Algorithm

• The size of K is $O(mn)$ in the worst case.

• We can further reduce the size of the proxy graph to $O(m)$

• Process the back edges according to a preorder visit of their destination vertex in the DFS tree

• Mark the discovery edges forming the cycles

• Stop adding edges to the proxy graph after the first marked edge is encountered.

• The resulting proxy graph is a forest!

• This algorithm runs in $O(n+m)$ time.
Example

- Back edges labeled according to the preorder visit of their destination vertex in the DFS tree

- Processing $e_1$

- Processing $e_2$
Example (contd.)

• DFS tree

• final proxy graph (a tree since the graph is biconnected)
Why Preorder?

• The order in which the back edges are processed is essential for the correctness of the algorithm.

• Using a different order ...

• ... yields a graph that provides incorrect information.
Try the Algorithm on this Graph!