If \((u, v) \in E(G)\), then \(u\) appears before \(v\) in the ordering.

- A linear ordering of all vertices of \(G = (V, E)\) is called a topological sort.

- Articulation points
- Topological sort
- Graph representation in Python

Recitation 12 - 1 April 2008
3. \text{return } L

2. As each vertex finishes, insert in front of $L$ \((v, W(v))\) \begin{itemize}
\item \((w, \text{link}(v))\)
\end{itemize}

1. Call DFS($G$) to compute $\text{LFN}(v)$ and $\text{VF}(v)$

\text{Topological Sort} (G)
Let $G = (V, E)$ be a connected, undirected graph.

- **Bridge** $e$ is an edge whose removal disconnects $G$.
- **Articulation Point** is a vertex whose removal disconnects $G$.

- **Biconnected Component** $C$ is a maximal set of edges, any two edges in the set.
- **Connectivity** in a connected simple cycle.

Diagram:

1. Connected graph
2. Bridge $e$ highlighted
3. Articulation points labeled 1 and 2
Consider root $r$, with only one children. If root is articulation point, then it has at least two children.

For $r$ to be root of $G$, it is an articulation point of $G$. If $r$ is not an articulation point of $G$, then it has only one children. If not, $G$ is not a connected component. Removing $r$ doesn't disconnect $G$.

Case 2: R has no back edge connecting back to itself.

Case 1: There is a back edge from a node in subtree pointing back to $r$. Removing a only removal one edge can't disconnect $G$.
We can prove this by contradiction. Assume an arbitrary node of $V$ 
has a back edge from $S$. Let $s$ be a proper ancestor of $V$. 
If $s$ is a root and $V$ is not, then $V$ is necessarily a descendant 
of a leaf $w$ having a child $s$ in it. Since there is no back edge from $s$ to any
child of $w$, we reach a contradiction.

Let $w$ be a non-root node of $G$. If $w$ is not a root, it has at least two children, 
and there can not be any cross edges.

If $w$ is a root, $w$ has at least two children, then it is an articulation point.

Running $V$ would disconnect those two (or more) components.

Therefore, the edges in the subtree $s$ can not be any cross edges.

The problem would be disconnected if we were to remove $V$.

Assume that $V$ is connected, we require $V$ to be a proper ancestor of $w$.

If $w$ is a root, $w$ has at least two children, then it is an articulation point. 

We can prove this by contradiction. Assume an arbitrary node of $V$. 
If $s$ is a root, then it has a back edge from $S$. Let $s$ be a proper ancestor of $V$. 
If $s$ is a root and $V$ is not, then $V$ is necessarily a descendant 
of a leaf $w$ having a child $s$ in it. Since there is no back edge from $s$ to any 
child of $w$, we reach a contradiction.
Compute $\text{Low}(V \setminus \{v\})$ for any vertex $v$ in $G$.

Let $D(v)$ be any back edge for some descendant $v$ of $G$.

$D(v)$ is a descendant of $v$. Hence, $v$ is an articulation point.

Removing $v$ will remove the tree edge and disconnect the graph. No back edge from subtree $G'$ of $v$ can only be ancestor of $v$.

If $G'$ is a subtree rooted at child of $v$ that has no back edge to a parent of $v$, then $v$ is still connected. Contradiction!
Initially call visit(
0)
\text{else}
\begin{align*}
\text{print } & \text{ if in articulation point } \\
\text{if } & (d(v) \neq 1 \text{ and } \text{low}(v) \leq \text{d}(v)) \\
\text{print } & \text{ if in articulation point } \\
\text{if } & (d(v) = 1 \text{ and } \text{d}(v) + 1 \leq \text{root}) \\
\text{for all nodes } & w \neq u, (w, v) \in E(G) \\
\text{visit } & (w, v) \\
\text{for some child } w \neq v \\
\text{if } & \text{low}(w) \leq \text{time} \\
\text{print } & \text{ if in articulation point } \\
\text{time} & = \text{time} + 1
\end{align*}
\text{(d) Show how to compute all articulation points in } O(n) \text{ time.}
Def: A bridge is an edge whose removal disconnects two articulation points or articulation points from an ancestor in G.

For any edge e = (u,v), e is a bridge if and only if every path between u and v in G uses e.

Show how to compute all bridges of G in O(n + e) time.

Proof: If e is a bridge, then e is in a cycle. If e is not a bridge, then e is not in any cycle.

If e is in a cycle, it disconnects u and v from each other. If e is not in a cycle, it disconnects u and v from each other.

If e is in a cycle, then e disconnects u and v from c. If e is not in a cycle, then e disconnects u and v from c.
\text{Component } C_i \text{ is connected if } |C_i| > 0.

\text{Component } C_i \text{ is connected if } |C_i| > 0.

\text{Component } C_i \text{ is connected if } |C_i| > 0.

\text{Component } C_i \text{ is connected if } |C_i| > 0.

\text{Component } C_i \text{ is connected if } |C_i| > 0.

\text{Component } C_i \text{ is connected if } |C_i| > 0.

\text{Component } C_i \text{ is connected if } |C_i| > 0.

\text{Component } C_i \text{ is connected if } |C_i| > 0.

\text{Component } C_i \text{ is connected if } |C_i| > 0.

\text{Component } C_i \text{ is connected if } |C_i| > 0.

\text{Component } C_i \text{ is connected if } |C_i| > 0.

\text{Component } C_i \text{ is connected if } |C_i| > 0.

\text{Component } C_i \text{ is connected if } |C_i| > 0.

\text{Component } C_i \text{ is connected if } |C_i| > 0.

\text{Component } C_i \text{ is connected if } |C_i| > 0.

\text{Component } C_i \text{ is connected if } |C_i| > 0.