Depth-first search

- Idea: instead of a queue to store gray nodes, use a stack.
- Algorithm visits new (white) vertices before dealing with older gray ones.
- Hence it tends to explore deeply first.
- More valuable than BFS, especially for directed graphs.
- We add a timestamp of colour changes to indicate when node turned gray \(d[u]\) and black \(f[u]\).
Pseudocode for DFS

DFS (G)
colour_all_vertices_white(); time<-0
while there is a white vertex s do
    DFS_visit(s)
done
DFS_visit(v)
colour v gray; time++; d[v]<-time
for each w adjacent to v do
    if w white then
        \( (v,w) \) tree edge
        DFS_visit(w)
    else
        \( (v,w) \) is non-tree edge
        colour v black; time++; f[v]<-time
Analysis of DFS

- Note stack is implicit here (stores parameters for recursive calls)
- "v on stack" means call to DFS-Visit(v) has not terminated
- DFS-Visit called once on every white node
- Each adjacency list run through once
- As with BFS, running time is $\Theta(|V| + |E|)$ or $\Theta(n + m)$. 
DFS on undirected graphs

- Let \((v, w)\) be an edge, \(d[v] < d[w]\)
- If \(w\) found first on \(v\)'s adjacency list
  - \(w\) must have been white
  - \((v, w)\) is a tree edge
- If \(v\) found first on \(w\)'s adjacency list
  - \(v\) is gray
  - \((v, w)\) is a back edge
Tree Edges and Back Edges

- **DFS on an undirected graph:**
  - For undirected graphs we have tree edges and all other edges not in the spanning tree are called back edges.
Absence of Cross Links

Again, consider DFS on an undirected graph:

- Let $u$ and $v$ be two vertices such that neither is a descendent of the other. Then there is no back edge between any descendent of $u$ and any descendent of $v$. 
The parenthesis theorem

Theorem 1

The intervals \([d[u], f[u]]\) and \([d[v], f[v]]\) are either nested (in which case the inner one is a descendant of the outer) or disjoint.
The parenthesis theorem

Proof: WLOG assume $d[u] < d[v]$,

- If $d[v] < f[u]$, $v$ was discovered while $u$ was gray (on the stack), so $v$ is a descendant of $u$ and $f[v] < f[u]$ (nested)

$$
\begin{array}{c}
d[u] & d[v] & 1 & \text{---} & 2n \\
f[v] & f[u] & \downarrow & & \\
\end{array}
$$
The parenthesis theorem

If \( f[u] < d[v] \), then intervals are disjoint

\[
1 \quad d[u] \quad f[u] \quad d[v] \quad f[v] \quad 2n
\]

Corollary 3

\( v \) is descendant of \( u \) if and only if

\[
d[u] < d[v] < f[v] < f[u]
\]
The white-path theorem

**Theorem 2**

$v$ is a descendant of $u$ if and only if at time $d[u]$, $v$ is reachable by a white path from $u$
The white-path theorem

Proof: If \( v \) a descendant of \( u \), by Corollary 3, every vertex on tree path from \( u \) to \( v \) has higher dvalue, so is white at time \( d[u] \).

If \( v \) reachable by white path at time \( d[u] \) but does not become descendant, assume every other vertex on path does (otherwise repeat argument for closest one to \( u \) that doesn’t).
The white-path theorem

Predecessor $w$ of $v$ in path is descendant of $u$, so $f[w] \leq f[u]$ ($w$ could be $u$)

$d[u] < d[v]$ ($v$ white when $u$ discovered)
  $< f[w]$ ($v$ must be discovered before $w$ finished)
  $\leq f[u]$ (by above)
The white-path theorem

\[ d[u] < d[v] < f[w] \leq f[u] \] (from last slide)

Since \([d[v], f[v]]\) nested inside \([d[u], f[u]]\), the parenthesis theorem says that \(v\) is a descendant of \(u\) (contradiction to the assumption that it was not).
Articulations

Definition:
- A node $v$ of a connected graph $G$ is an **articulation** point (also called a cut vertex) if the removal of $v$ and all its incident edges causes $G$ to become disconnected.

Motivation for articulations:
- Articulations are important in communication networks.
- In traffic flows they identify places which will stop traffic between two areas of a city if they become blocked.
Finding Articulations

- Problem:
  - Given any graph $G = (V, E)$, find all the articulation points.

- Possible strategy:
  - For all vertices $v$ in $V$:
    - Remove $v$ and its incident edges
    - Test connectivity using a DFS.
  - Execution time: $\Theta(n(n + m))$.
    - Can we do better?
Finding Articulation Points

- A DFS tree can be used to discover articulation points in $\Theta(n + m)$ time.
  - We start with a program that computes a DFS tree labeling the vertices with their discovery times.
  - We also compute a function called $low(v)$ that can be used to characterize each vertex as an articulation or non-articulation point.
    - The root of the DFS tree (the root has a $d[\ ]$ value of 1) will be treated as a special case:
Finding Articulation Points

- The root of the DFS tree is an articulation point if and only if it has two children.
  - Suppose the root has two or more children.
    ★ Recall that the back edges never link the vertices in two different subtrees.
    ★ So, the subtrees are only linked through the root vertex and if it is removed we will get two or more connected components (i.e. the root is an articulation point).
  - Suppose the root is an articulation point.
    ★ This means that its removal would produce two or more connected components each previously connected to this root vertex.
    ★ So, the root has two or more children.
Computation of $\text{low}(v)$

- We need another function defined on vertices: This quantity will be used in our articulation finding algorithm:
  $$\text{low}(v) = \min\{d[v], d[w] : (u, w) \text{ is a back edge for some descendent } u \text{ of } v\}$$

- So, $\text{low}(v)$ is the discovery time of the vertex closest to the root and reachable from $v$ by following zero or more edges downward, and then at most one back edge.
Finding Articulation Points

- For non-root vertices we have a different test.
  - Suppose $v$ is a non-root vertex of the DFS tree $T$. Then $v$ is an articulation point of $G$ if and only if there is a child $w$ of $v$ in $T$ with $low(w) \geq d[v]$.
  - Sufficiency: Assume such a child $w$ exists.
    - There is no descendent vertex of $v$ that has a back edge going “above” vertex $v$.
    - Also, there is no cross link from a descendent of $v$ to any other subtree.
    - So, when $v$ is removed the subtree with $w$ as its root will be disconnected from the rest of the graph.
Finding Articulation Points

- Necessity: Assume no such child \( w \) exists.
  - In this case all children of \( v \) have a descendent with a back edge going to an ancestor of \( v \).
  - When \( v \) is removed each of the children of \( v \) will still be connected to some vertex on the path going from the root to the vertex.
  - The graph stays connected, and so \( v \) would not be an articulation point in this case.
Finding Articulation Points Pseudocode

function dfs-visit(v)
    status[v] := gray; time := time+1; d[v] := time;
    low[v] := d[v];
    for each w in out(v)
        if status[w] = white
            //--- (v,w) is a TREE edge
            dfs-visit(w); // low[w] is now computed!
            if low[w] >= d[v] then
                record that vertex v is an articulation
                if low[w] < low[v] then low[v] := low[w];
                else if w is not the parent of v then
                    //--- (v,w) is a BACK edge
                    if d[w] < low[v] then low[v] := d[w];
            status[v] := black;