DFS(v) – Recursive version

Global Initialization:

for all nodes v, v.dfs# = -1       // mark v "undiscovered"
dfscounter = 0
for v = 1 to n do
    if state(v) != fully-explored then
        DFS(v):
        v.dfs# = dfscounter++          // v "discovered", number it
        Mark v `discovered`.
        for each edge (v,x)
            if (x.dfs# == -1)           // (x previously undiscovered)
                DFS(x)
            else …
        Mark v “fully-explored”
Kinds of edges – DFS on directed graphs

Edge \((u,v)\)

- **Tree**
  
  \[
  [u \ [v \ v] \ u]
  \]

- **Forward**
  
  \[
  [u \ [v \ v] \ u]
  \]

- **Cross**
  
  \[
  [v \ v] \ [u \ u]
  \]

- **Back**
  
  \[
  [v \ [u \ u] \ v]
  \]
Topological Sort using DFS

Global Initialization:

```
for all nodes v, v.dfs# = -1  // mark v "undiscovered"
dfscounter = 0
current_label = n
for v = 1 to n do
  if state(v) != fully-explored then
    DFS(v):
```

```
DFS(v):
  v.dfs# = dfscounter++  // v "discovered", number it
  Mark v "discovered".
  for each edge (v,x)
    if (x.dfs# == -1)  // (x previously undiscovered)
      DFS(x)
    else               // add check for cycle if needed
      Mark v "fully-explored"
  f(v) = current_label  // f(v) values give the topological order
  current_label --;
```
Analysis

Running time $O(n+m)$

Correctness: Need to show that:

if \((u,v)\) is an edge then $f(u) < f(v)$

Case 1: $\text{DFS}(u)$ called before $\text{DFS}(v)$, so $\text{DFS}(v)$ finishes first, which means $f(v) > f(u)$. 

Case 2: $\text{DFS}(v)$ called before $\text{DFS}(u)$. But there cannot be a directed path from $v$ to $u$, so recursive call to $\text{DFS}(v)$ will finish before recursive call to $\text{DFS}(u)$ starts, so $f(v) > f(u)$
A simple problem on trees

Given: tree T, a value \( L(v) \) defined for every vertex \( v \) in T

Goal: find \( M(v) \), the min value of \( L(v) \) anywhere in the subtree rooted at \( v \) (including \( v \) itself).

How? Depth first search, using:

\[
M(v) = \begin{cases} 
L(v) & \text{if } v \text{ is a leaf} \\
\min(L(v), \min_{w \text{ a child of } v} M(w)) & \text{otherwise}
\end{cases}
\]
DFS(v) – Recursive version

Global Initialization:
for all nodes v, v.dfs# = -1 // mark v "undiscovered"
dfscounter = 0 // (global variable)
DFS(s); // start DFS at node s;

DFS(v)
v.dfs# = dfscounter++ // v "discovered", number it
for each edge (v,x)
  if (x.dfs# = -1) // tree edge (x previously undiscovered)
    DFS(x)
Application: Articulation Points

A node in an undirected graph is an \textbf{articulation point} iff removing it disconnects the graph.

articulation points represent vulnerabilities in a network – single points whose failure would split the network into 2 or more disconnected components.
Identifying key proteins on the anthrax predicted network

Articulation point proteins
Articulation Points

articulation point
iff its removal disconnects the graph
Articulation Points
Simple Case: Artic. Pts in a tree

Which nodes in a rooted tree are articulation points?
Simple Case: Artic. Pts in a tree

Leaves – never articulation points
Internal nodes – always articulation points
Root – articulation point if and only if two or more children

Non-tree: extra edges remove some articulation points (which ones?)
Recall: all edges either tree edges or back edges in DFS on undirected graph

Consider edge \((u,v)\).

If \(u\) discovered first, then edge \((u,v)\) will be explored before DFS\((u)\) completes.

If at the time it is explored \(v\) is undiscovered, the edge will become a tree edge.

If \(v\) is already discovered, then since DFS\((v)\) was called after DFS\((u)\), it completes before DFS\((u)\) completes,

So \(v\) is a descendent of \(u\).
Recall: all edges either tree edges or back edges in DFS on undirected graph

If u is an ancestor of v, then

dfs# of u is lower than dfs# of v
Simple Case: Artic. Pts in a tree

Leaves – never articulation points
Internal nodes – always articulation points
Root – articulation point if and only if two or more children

Non-tree: extra edges remove some articulation points (which ones?)
Articulation Points from DFS

Root node is an articulation point iff …. 

Leaf is never an articulation point

non-leaf, non-root node \( u \) is an articulation point
Articulation Points from DFS

Root node is an articulation point iff it has more than one child.

Leaf is never an articulation point.

Non-leaf, non-root node $u$ is an articulation point

$\exists$ some child $y$ of $u$ s.t.
no non-tree edge goes above $u$ from $y$ or below

If removal of $u$ does NOT separate $x$, there must be an exit from $x$'s subtree. How?
Via back edge.
Articulation Points: the "LOW" function

Definition: $\text{LOW}(v)$ is the lowest $\text{dfs}\#$ of any vertex that is either in the $\text{dfs}$ subtree rooted at $v$ (including $v$ itself) or connected to a vertex in that subtree by a back edge.
LOW(v) is the lowest dfs# of any vertex that is either in the dfs subtree rooted at v (including v itself) or connected to a vertex in that subtree by a back edge.
Articulation Point

**Diagram:**

- **Vertices:** A, B, C, D, E, F, G, H, I, J, K, L, M
- **Articulation Points:** B, D, H, I
- **DFS Numbers:**
  - A: 1
  - B: 2
  - C: 3
  - D: 4
  - E: 8
  - F: 5
  - G: 9
  - H: 10
  - I: 6
  - J: 11
  - K: 7
  - L: 12
  - M: 13
- **Low Values:**
  - A: 1
  - B: 1
  - C: 1
  - D: 3
  - E: 1
  - F: 3
  - G: 9
  - H: 1
  - I: 3
  - J: 10
  - K: 3
  - L: 10
  - M: 13
Articulation Points: the "LOW" function

Definition: LOW(v) is the lowest dfs# of any vertex that is either in the dfs subtree rooted at v (including v itself) or connected to a vertex in that subtree by a back edge.

v articulation point iff…
Articulation Points:
the "LOW" function

Definition: \( \text{LOW}(v) \) is the lowest dfs\# of any vertex that is either in the dfs subtree rooted at \( v \) (including \( v \) itself) or connected to a vertex in that subtree by a back edge.

\( v \) (non-root) articulation point iff some child \( x \) of \( v \) has \( \text{LOW}(x) \geq \text{dfs}\#(v) \)
Articulation Points: the "LOW" function

Definition: LOW(v) is the lowest dfs# of any vertex that is either in the dfs subtree rooted at v (including v itself) or connected to a vertex in that subtree by a back edge.

v (nonroot) articulation point iff some child x of v has \( \text{LOW}(x) \geq \text{dfs#}(v) \)

\[
\text{LOW}(v) = \min \left( \{ \text{dfs#}(v) \} \cup \{ \text{LOW}(w) \mid w \text{ a child of } v \} \cup \{ \text{dfs#}(x) \mid \{v, x\} \text{ is a back edge from } v \} \right)
\]
DFS(v) for Finding Articulation Points

Global initialization: v.dfs# = -1 for all v.

DFS(v)

v.dfs# = dfscounter++

v.low = v.dfs# // initialization

for each edge {v,x}

if (x.dfs# == -1) // x is undiscovered

    DFS(x)

    v.low = min(v.low, x.low)

if (x.low >= v.dfs#)

    print "v is art. pt., separating x"

else if (x is not v's parent)

    v.low = min(v.low, x.dfs#)

Equiv: "if( {v,x} is a back edge)"

Why?
Summary

Graphs – abstract relationships among pairs of objects
Terminology – node/vertex/vertices, edges, paths, multi-edges, self-loops, connected
Representation – edge list, adjacency matrix
Nodes vs Edges – $m = O(n^2)$, often less
BFS – Layers, queue, shortest paths, all edges go to same or adjacent layer
DFS – recursion/stack; all edges ancestor/descendant
Algorithms – connected components, bipartiteness, topological sort, articulation points