To find the SCC's of a digraph $G$ do:

- call $DFS(G)$, as vertices finish, push them onto a stack.
- compute the transpose $G^T$ of $G$. (reverse all directed edges.)
- call $DFS(G^T)$, processing vertices in main loop at $DFS$ by decreasing times from first call, i.e. pop vertices off the stack.
Thus (22.16 P. 556)
when this process is completed
the trees in the DFS forest
from 2 nd call to DFS span the
SCCs of G.
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>
Strong components of $G$ (and $G^T$) are

$C_1 = \{1, 5, 2\}$

$C_2 = \{3, 4\}$

$C_3 = \{7, 6\}$

$C_4 = \{8\}$
\[ \text{Defn} \]

The component graph (also condensation graph) \( G^\text{sc} \) of \( G \) has \( V(G^\text{sc}) = \{ \text{the strong components of } G \} \), and there exists an edge from \( C_i \) to \( C_j \) in \( G^\text{sc} \) iff there exists \( x \in C_i \) and \( y \in C_j \) s.t. \((x, y) \in E(G)\).
\[ C_{\text{scc}} : \]

\[ \begin{align*}
1, 2, 5 & \rightarrow & 3, 4 \\
6, 7 & \rightarrow & 8 \\
\end{align*} \]

\text{Note: } \quad C_{\text{scc}} \ \text{is necessarily acyclic.}

\text{Topological sort:}

\[ \begin{align*}
C_1 & \rightarrow & C_2 & \rightarrow & C_3 & \rightarrow & C_4 \\
1, 2, 5 & \rightarrow & 3, 4 & \rightarrow & 6, 7 & \rightarrow & 8 \\
\end{align*} \]
Stack: bottom

2 5 1 | 4 3 | 6 7 | 8
top

Parent:

n | n | n | n

Output:

Component 1: 1 5 2

... 2: 3 4

... 3: 7 6

... 4: 8

See Examples: S 4 and out 4
Prototype for DFS(·, ·)

void DFS(Graph G, List S);

call it on S: 1 2 3 ... n

DFS(G, S)

DFS(G^T, S)

use status of S to determine a topological sort of G.
**Appendix B.5 Trees**

**Defn**

A rooted tree is a tree in which one vertex is distinguished as the root.

![Diagram of a rooted tree]

**Example**

- $x$ is parent of $z$.
- $y, z$ are children of $x$.

**Height**

- $y$: height = 3
- $x$: depth = 0
- $z$: depth = 2
- $r$: depth = 3

**Subtree rooted at $x$**

- Height($x$) = 2
A vertex with no children is a leaf.

A non-leaf is an internal node.

Note: 'node' = 'vertex'.

Defn. The height of a rooted tree is the depth of its deepest leaf.

Defn. The height of a node \( x \) in the height of subtree rooted at \( x \).
**Definition**

A binary tree is a rooted tree in which every node has at most 2 children, identified as left child and right child.

**Example**

Same rooted tree, but different binary trees.
Recursive definition of height of a binary tree $T$

\[
h(T) = \begin{cases} 
-\infty & n = 0 \\ 
0 & n = 1 \\ 
1 + \max(h(L), h(R)) & n \geq 2
\end{cases}
\]

where $n = \# \text{nodes in } T$, and $L, R$ are the left and right subtrees, respectively.
Exercise (Problem 3.5-4)

Show that any binary tree $T$ with $n$ nodes satisfies

$$h(T) \geq \lceil \lg(n) \rceil.$$

**Hint:** Use strong induction on $n$, beginning at $n=1$

**Hint:** First show

$$\lceil \lg(2k+1) \rceil = \lceil \lg(2k) \rceil$$

for any $k \in \mathbb{Z}^+$. 
**Definition**

A complete binary tree (CBT) is a B.T. in which all leaves have same depth and all internal nodes have 2 children.

**Example**

\[ \text{(# nodes at depth } d) = 2^d \]