Tree Terminology

Trees

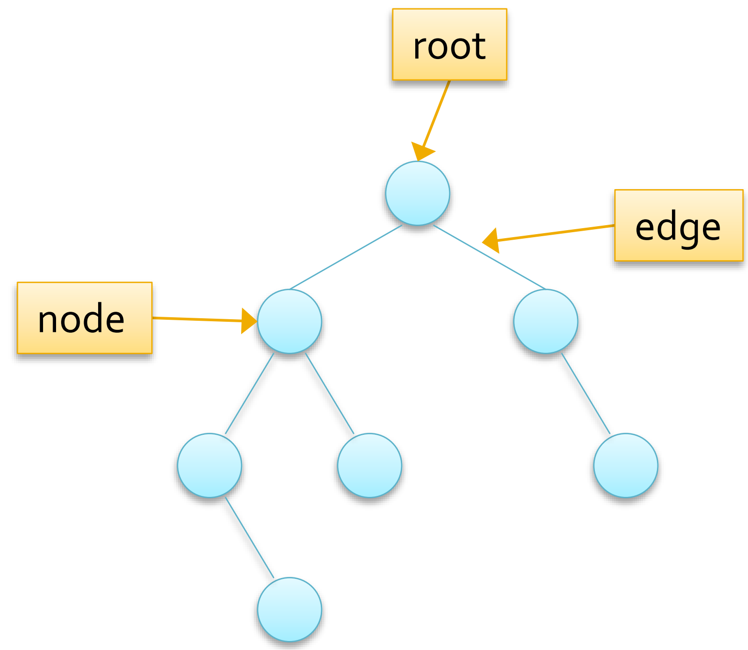
Topics

- Tree terminology
- Tree traversals

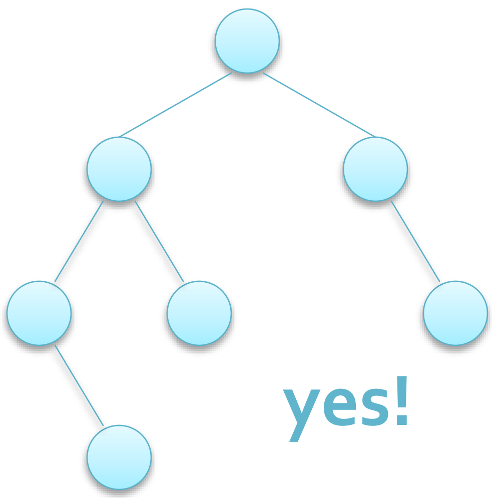
Tree Terminology

Trees

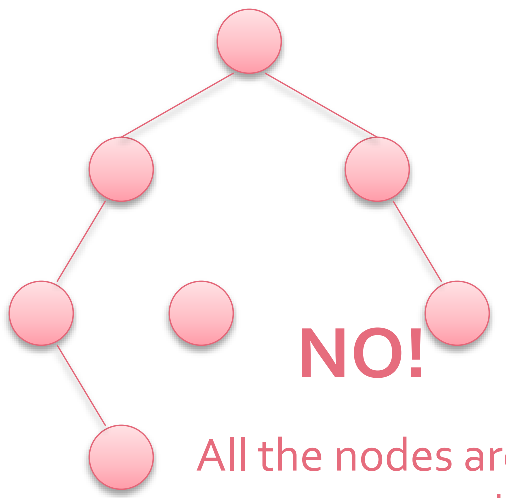
- A set of nodes (or vertices) with a single starting point
  - called the *root*
- Each node is connected by an *edge* to another node
- A tree is a connected graph
  - There is a path to every node in the tree
  - A tree has one less edge than the number of nodes



Is it a Tree?

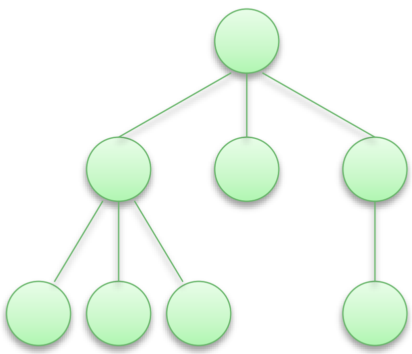


yes!

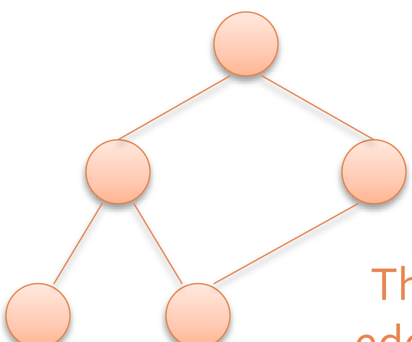


NO!

All the nodes are not connected

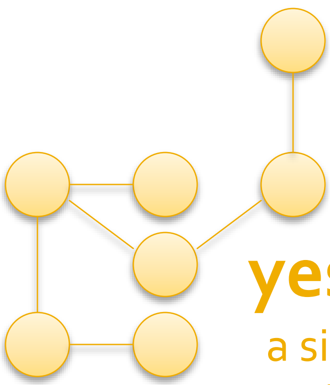


yes! (but not a *binary* tree)



NO!

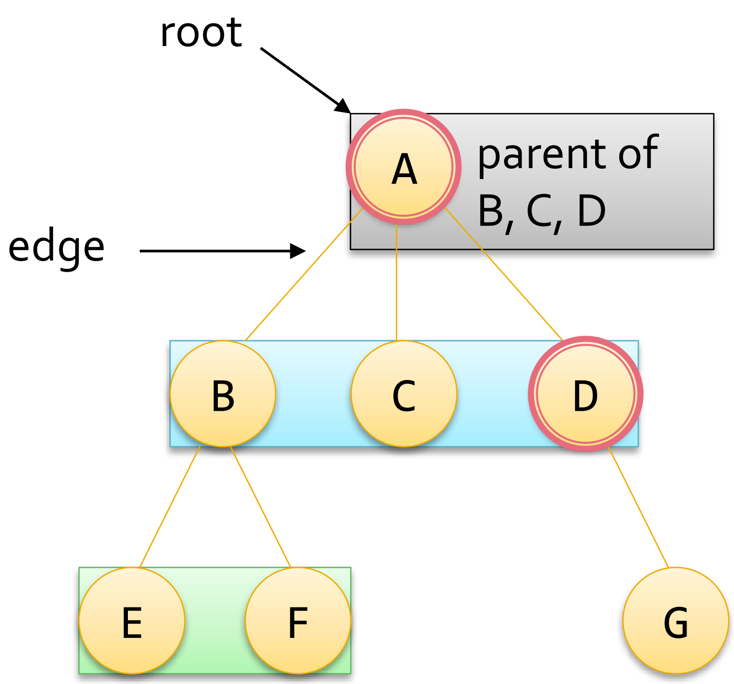
There is an extra edge (5 nodes and 5 edges)



yes! (it's actually a similar graph to the blue one)

Tree Relationships

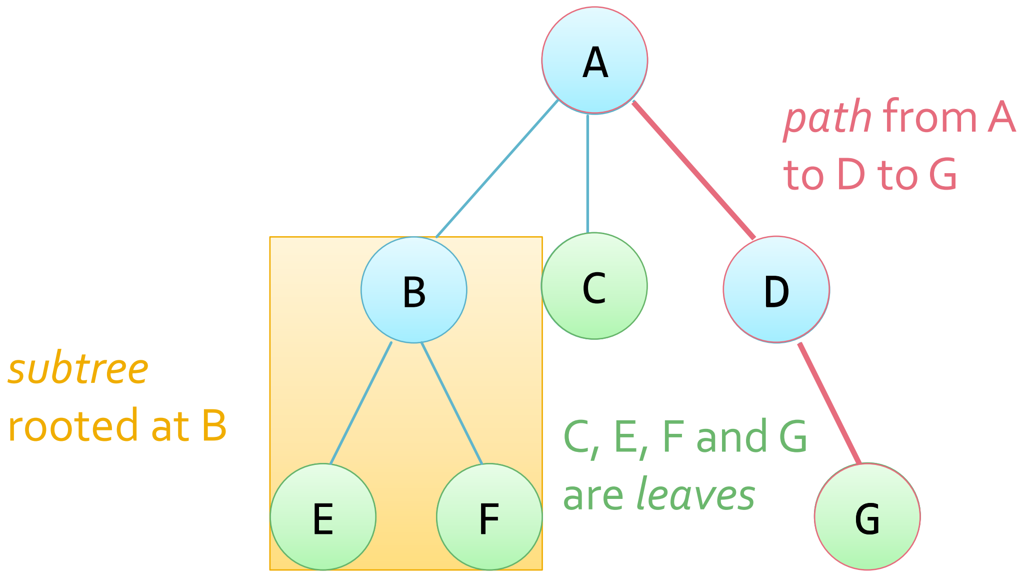
- Node *v* is said to be a *child* of *u*, and *u* the *parent* of *v* if
  - There is an edge between the nodes *u* and *v*, and
  - *u* is above *v* in the tree,
- This relationship can be generalized
  - E and F are *descendants* of A
  - D and A are *ancestors* of G
  - B, C and D are *siblings*
  - F and G are?



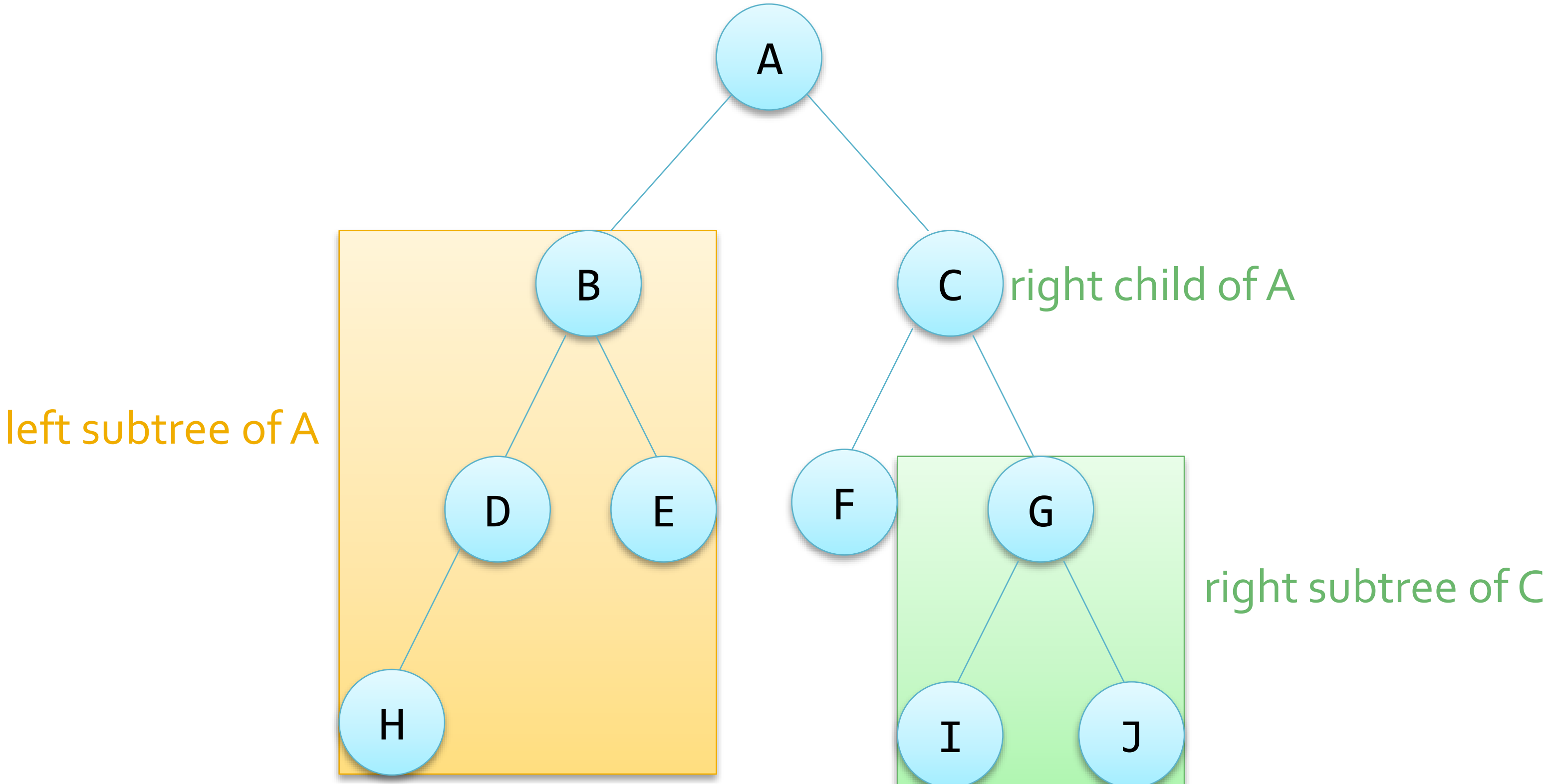
More Tree Terminology

- A *leaf* is a node with no children
- A *path* is a sequence of nodes $v_1 \ldots v_n$
  - where v_i is a parent of v_{i+1} $(1 \le i \le n)$
- A *subtree* is any node in the tree along with all of its descendants
- A *binary tree* is a tree with at most two children per node
  - The children are referred to as *left* and *right*
  - We can also refer to left and right subtrees

Tree Terminology Example



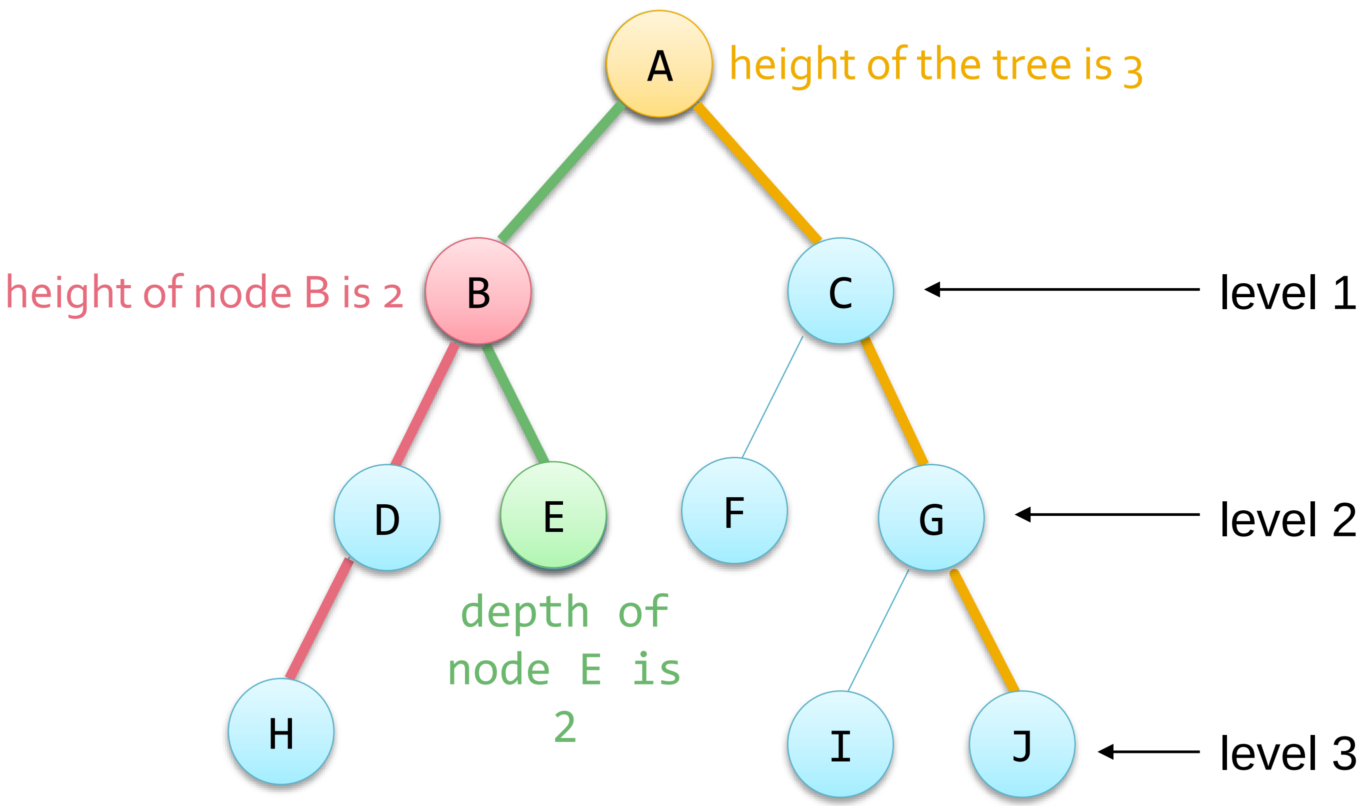
Binary Tree Terminology



Measuring Trees

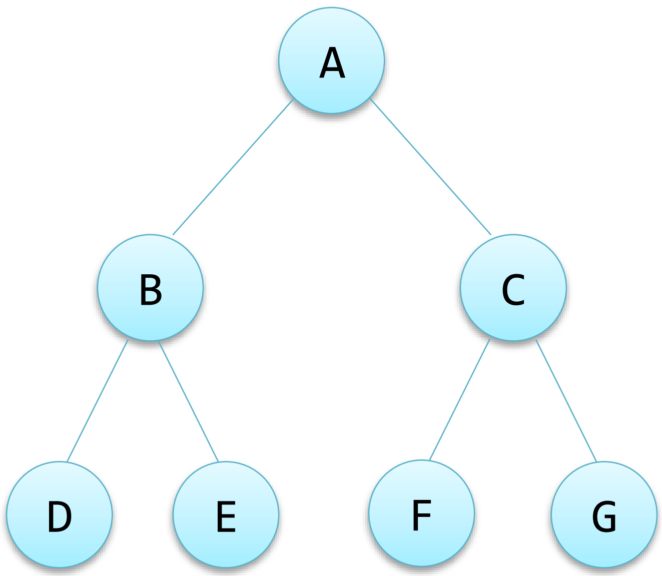
- The *height* of a node *v* is the length of the longest path from *v* to a leaf
  - The height of the tree is the height of the root
- The *depth* of a node *v* is the length of the path from *v* to the root
  - This is also referred to as the *level* of a node
- Note that there is a slightly different formulation of the height of a tree
  - Where the height of a tree is said to be the number of different *levels* of nodes in the tree (including the root)

Height of a Binary Tree



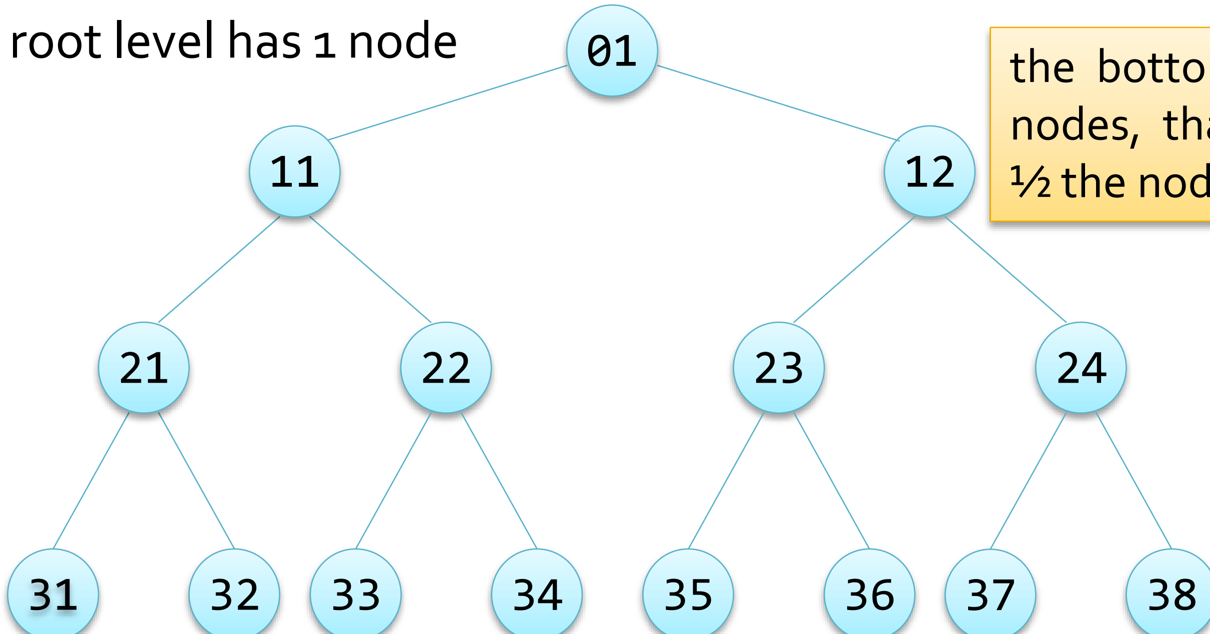
Perfect Binary Trees

- A binary tree is *perfect*, if
  - No node has only one child
  - And all the leaves have the same depth
- A perfect binary tree of height h has
  - $2^{h+1} - 1$ nodes, of which 2^h are leaves
- Perfect trees are also *complete*



Nodes in a Perfect Tree

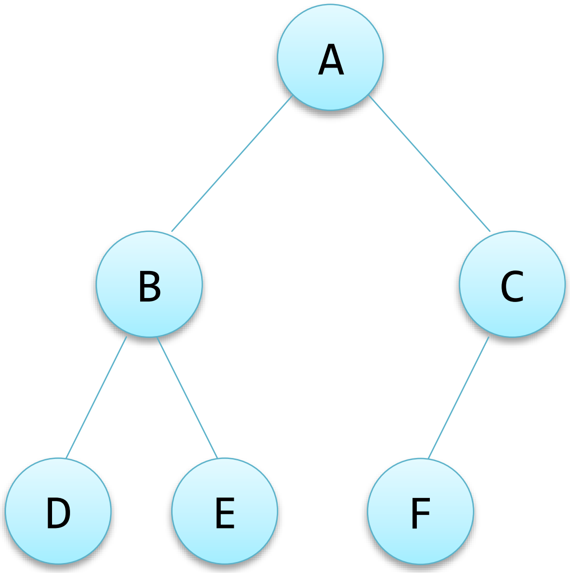
- Each level doubles the number of nodes
  - Level 1 has 2 nodes (2^1)
  - Level 2 has 4 nodes (2^2) or 2 times the number in Level 1
- Therefore a tree with h levels has $2^{h+1} - 1$ nodes
  - The root level has 1 node



the bottom level has 2^h nodes, that is, just over ½ the nodes are leaves

Complete Binary Trees

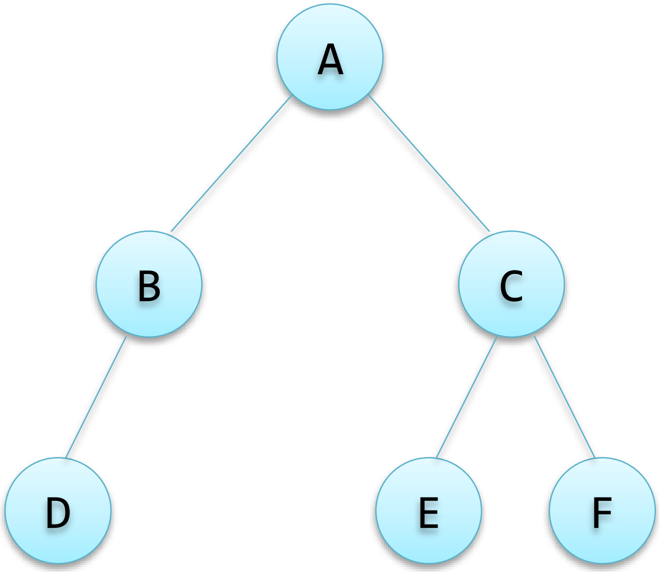
- A binary tree is *complete* if
  - The leaves are on at most two different levels,
  - The second to bottom level is completely filled in and
  - The leaves on the bottom level are as far to the left as possible

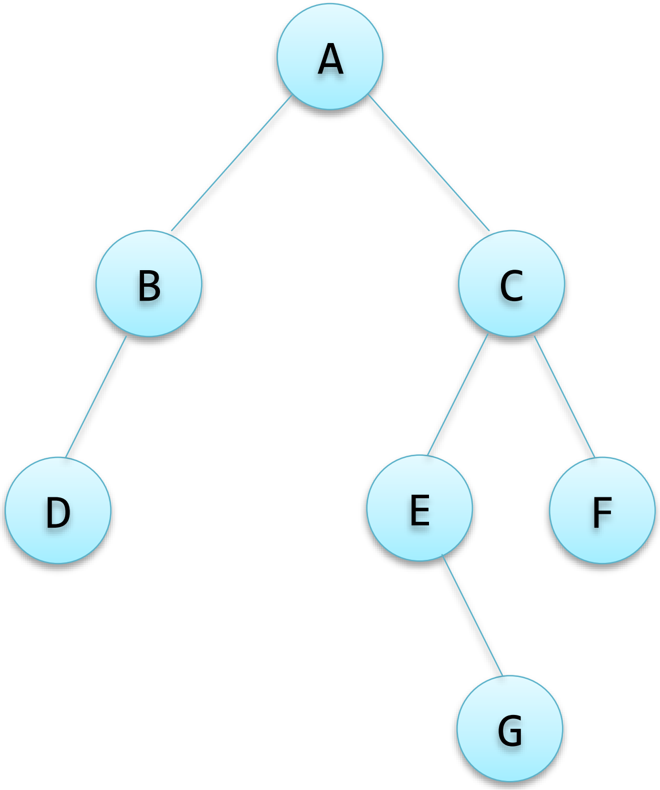


Balanced Binary Trees

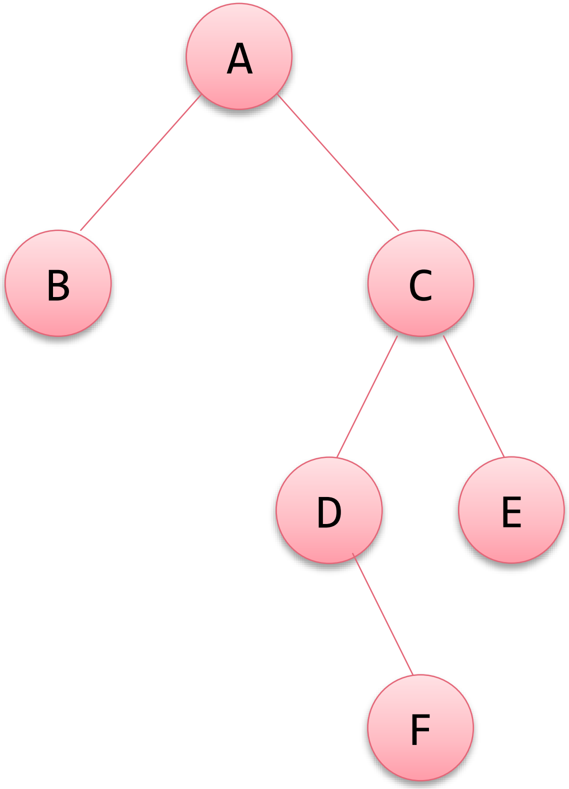
- A binary tree is *balanced* if
  - Leaves are all about the same distance from the root
  - The exact specification varies
- Sometimes trees are balanced by comparing the height of nodes
  - e.g. the height of a node's right subtree is at most one different from the height of its left subtree
- Sometimes a tree's height is compared to the number of nodes
  - e.g. red-black trees

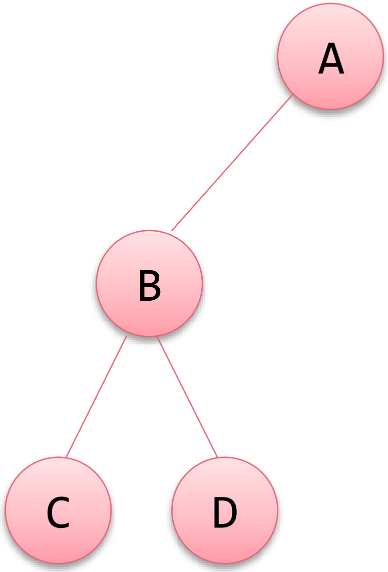
Balanced Binary Trees





Unbalanced Binary Trees





Tree Traversals

Binary Tree Traversals

- A traversal algorithm for a binary tree visits each node in the tree
  - Typically, it will do something while visiting each node!
- Traversal algorithms are naturally recursive
- There are three traversal methods
  - Inorder
  - Preorder
  - Postorder

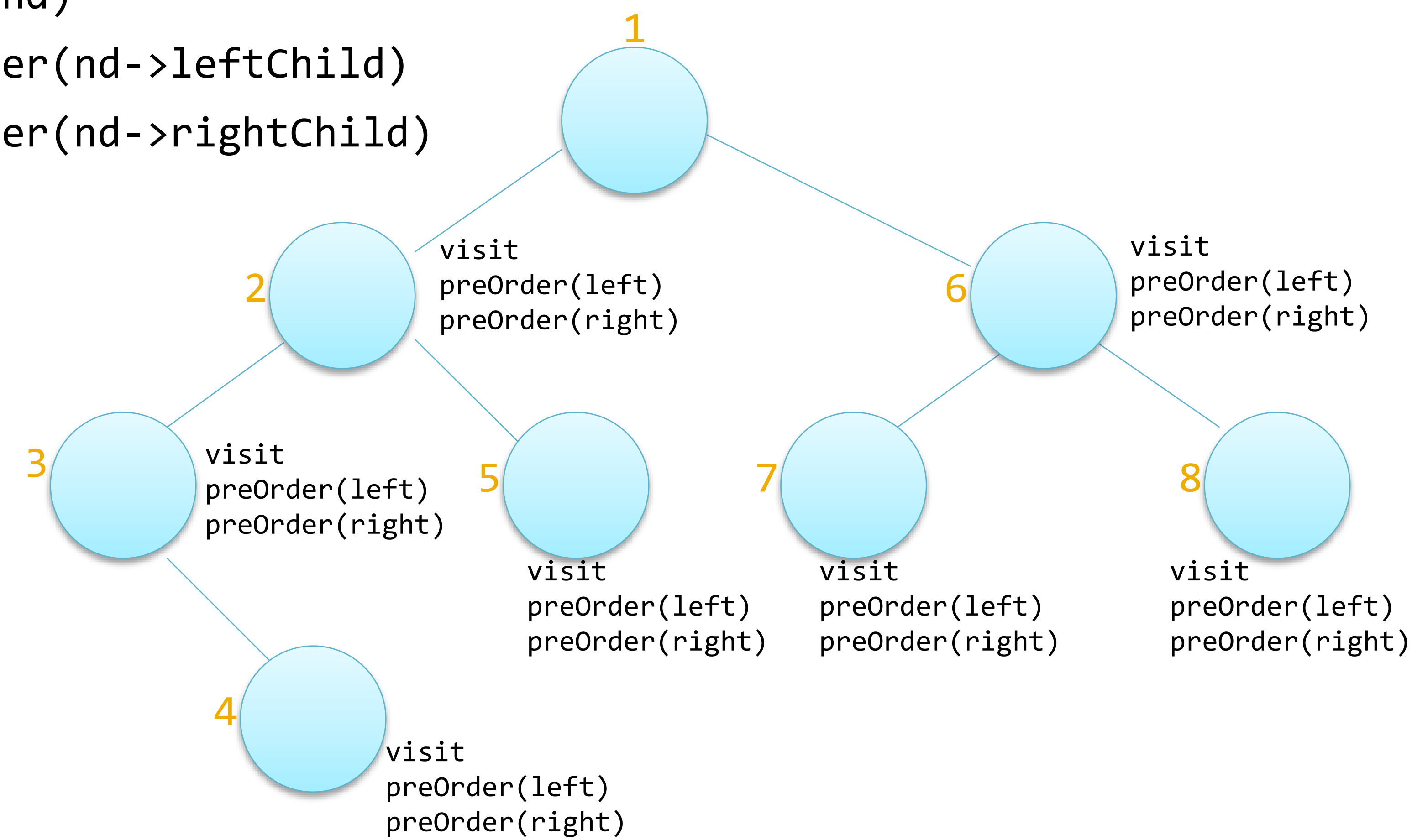
InOrder Traversal Algorithm

```
inOrder(Node* nd) {
    if (nd != NULL) {
        inOrder(nd->leftChild);
        visit(nd);
        inOrder(nd->rightChild);
    }
}
```

The visit function would do whatever the purpose of the traversal is, for example print the data in the node

PreOrder Traversal

```
visit(nd)
preOrder(nd->leftChild)
preOrder(nd->rightChild)
```



PostOrder Traversal

```
postOrder(nd->leftChild)
postOrder(nd->rightChild)
visit(nd)
```