Maps / Set

Association between a key → corresponding value

Put – add something to the map

Get – retrieve a value from the map based on its key

A set only has keys, not values.

{1, 3, 5, 7} = {3, 1, 7, 5}

– is 3 a member of this set or not? (yes)

– Is 4 a member of this set? (No)

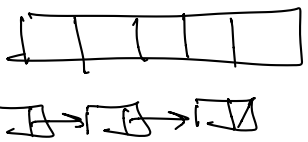
More efficient implementations of Maps / Sets.

Arraylists / LL → O(n) (worst case)

"Tree" → O(log n)

Trees

So far all of our data structures have been linear.



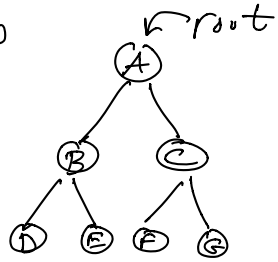
→ Hierarchical data structure

"Node"

Most important – root node – usually drawn @ top

Every node in a tree may have children (child nodes)

Every node in a tree has a parent node (except the root)



What is the parent of C?

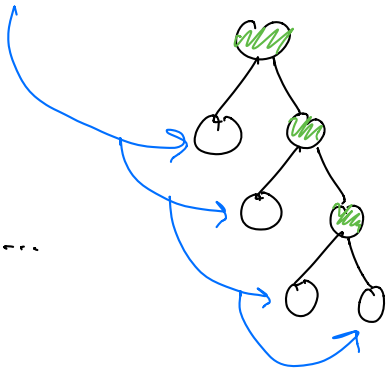
"A"

What are the children of B?

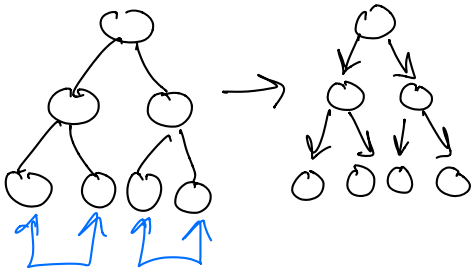
"D & E"

Nodes that have no children are called **leaves** (leaf nodes)

A node w/ at least one child is called an **internal node.**



- Descendant: children + children's children + ...
- Ancestor: parent + parent's parent + ...
- Sibling: nodes that share their parent



How big is a tree?

Depth: The depth of a node is the number of edges between that node & the root.

Height: The height of a node = # of edges on the longest path from the node to any leaf.

Height of the tree: The height of the root node.

Binary Tree: Every node has at most [2] children.

```
class Node {
    int/double/Object _ data;
    Node left;
    Node right;
}
```