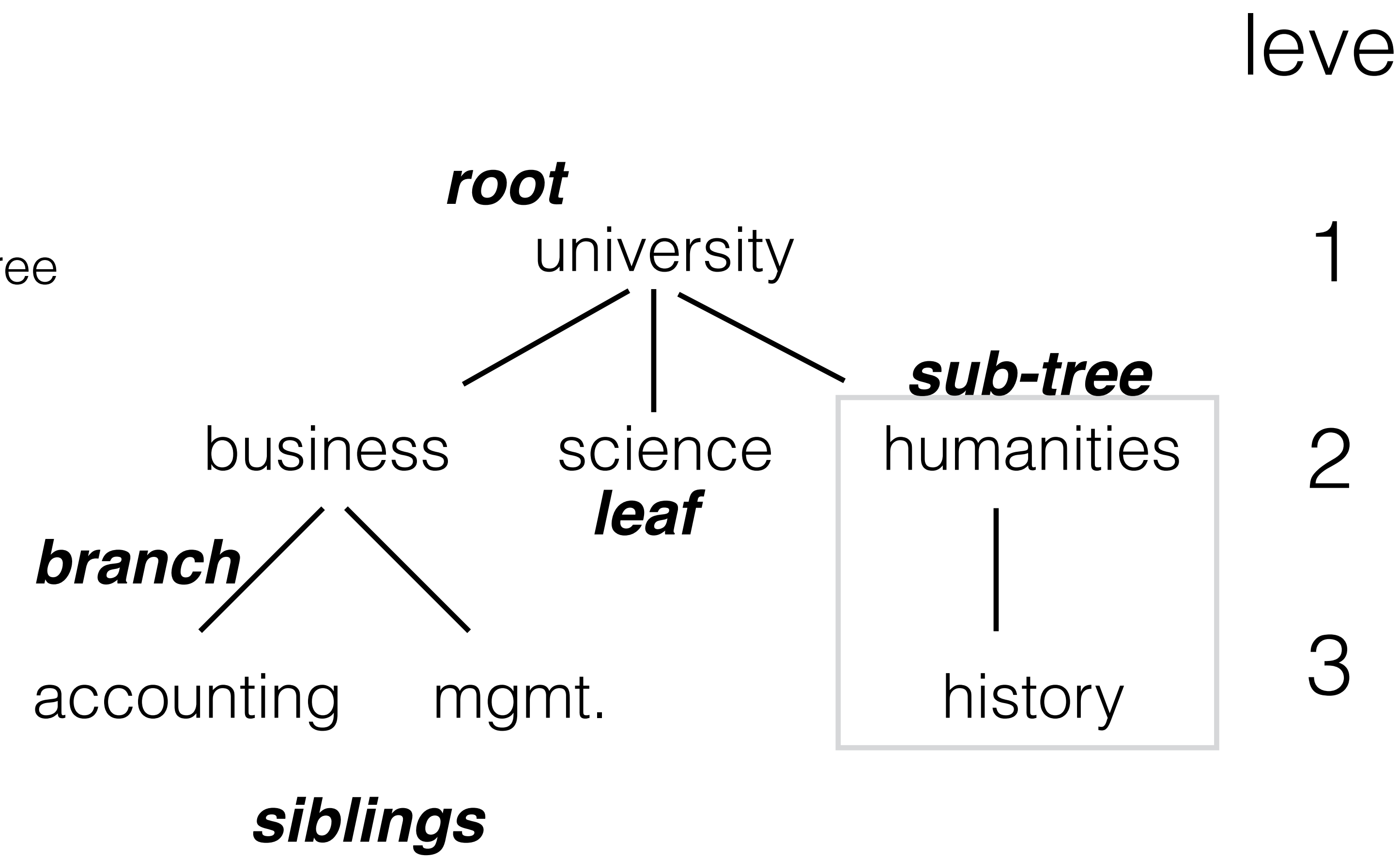
Search Algorithms, Trees, and Graphs Part I

Anton Gerdelan <gerdela@scss.tcd.ie>

General Trees

- **Node** - item in the tree
- **Branch** - link connecting nodes
- **Sub-tree** - part of tree that is also a tree
- **Root** - the node with no **parent**
- **Leaf** - nodes with no **children**
- **Siblings** - children of same parent
- **Levels** - Number of rows
- **Null tree** - nothing in tree
- **Tree** is either
    - a null tree
    - a root with several sub-trees



Typical Tree Operations

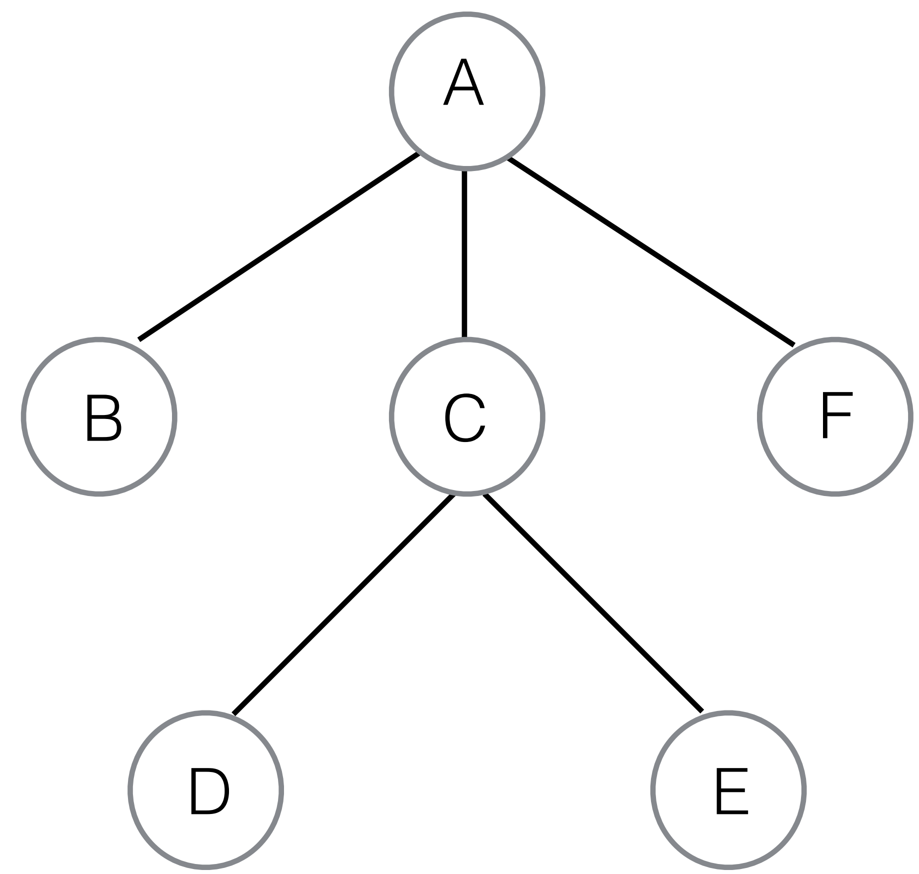
- **`data( Node* N )`** - return value of data in a node
- **`parent( Node* N )`** - return pointer to parent
- **`leftmost_child( Node* N )`**
    - return pointer to furthest left child of N
- **`right_sibling( Node* N )`**
- **`insert_child( Node* N, data )`**
    - make a new node with data and make it leftmost child of N
- **`is_empty()`** - return true if tree empty
- *How many pointers will your tree Node have?*

Searching for Data in a Tree

- An algorithm that visits each node once - **tree traversal** - and compares contents to sought value
- traversal ops: **V** - **visit** / look at node, **L** - **left** sub-tree, **R** - **right** sub-tree
- **Pre-order**: **V, L to R**
    1. look at data in root
    2. recurse all subtrees from left to right
- **In-order**: **L, V to R**
    1. recurse leftmost subtree
    2. look at root of current sub-tree
    3. recurse remaining subtrees to rightmost subtree
- **Post-order**: **L to R, V**
    1. recurse all subtrees from left to right
    2. look at root

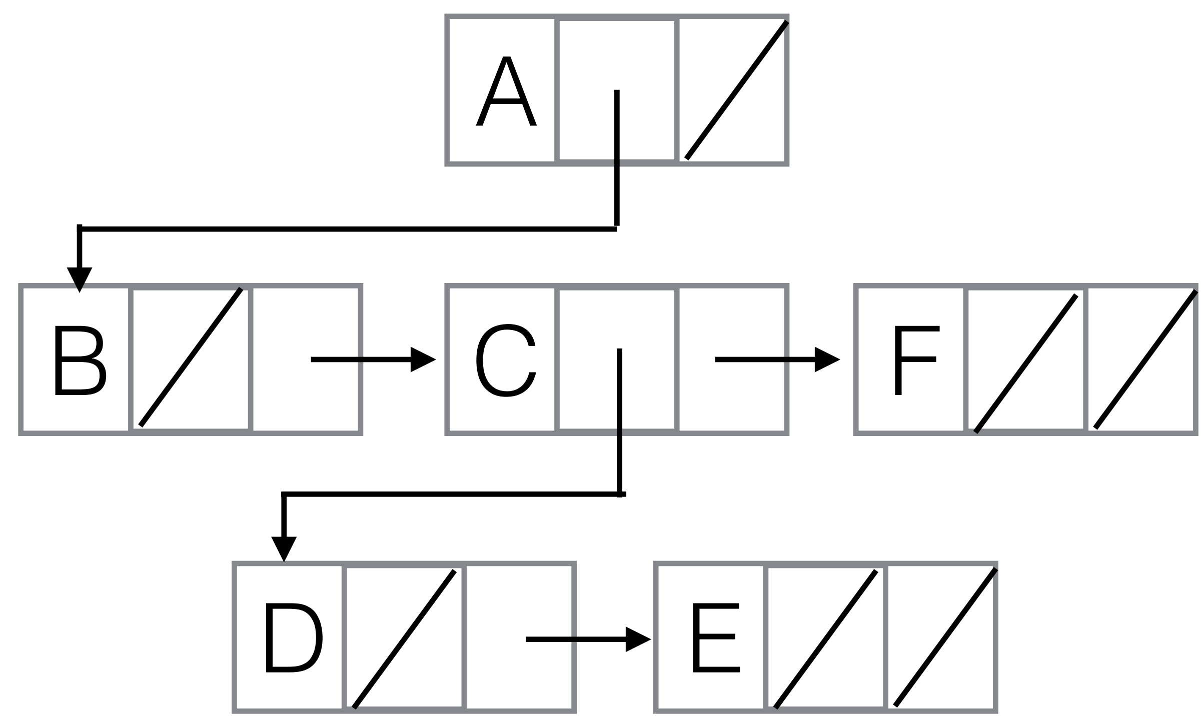
Traversal

- Using <u>recursion</u>
- Pre-Order: **VL to R**
  - A B C D E F
- In-Order: **LV to R**
  - B A D C E F
- Post-Order: **L to RV**
  - B D E C F A
- Some traversals more useful than others depending on situation
  - recall BSP



"**General Tree**" Implementation

- Q. Remember how to implement a tree using arrays?
- <u>Any</u> number of children
- Using pointers a tree is similar to a linked list
    - leftmost child pointer
    - right <u>sibling</u> pointer
- A **limited tree** with fixed number of children would be easier



Binary Trees

- 2 branches or less per node
- traversals only have 2 sub-trees
- VLR (pre order), LVR (in-order), LRV (post-order)
- writing a *function* to add a new node is a little tricky (as with linked lists)
    - tutorial?

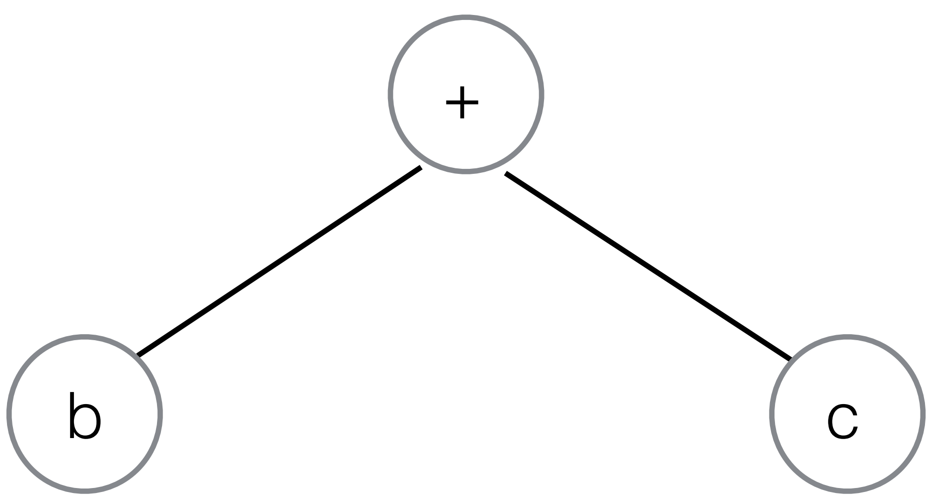
```
struct Tree_Node;
struct Tree_Node {
    char data;
    Tree_Node *left, *right;
};

Tree_Node *root = (Tree_Node*)malloc(
    sizeof(Tree_Node));
root->data = 'A';
root->left = NULL;
root->right = NULL;
```

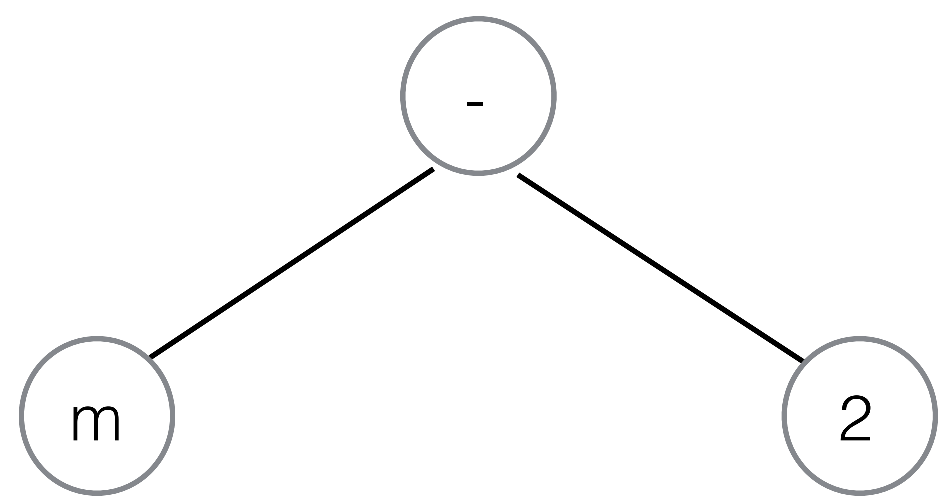
Arithmetic Trees

- construct tree to represent arithmetic expression
  $a * (b + c) / (m - 2)$

  - set up sub-tree for each set of brackets



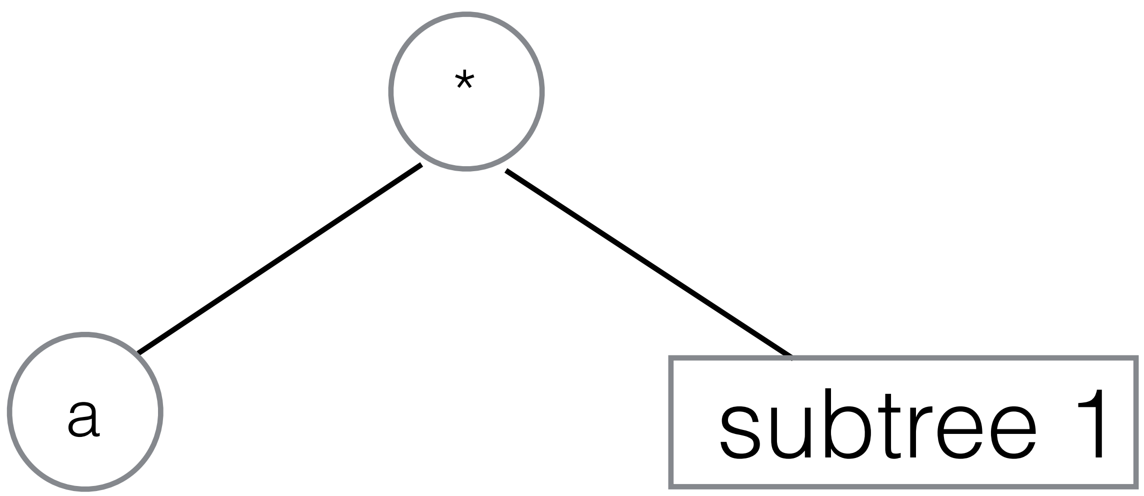
subtree 1



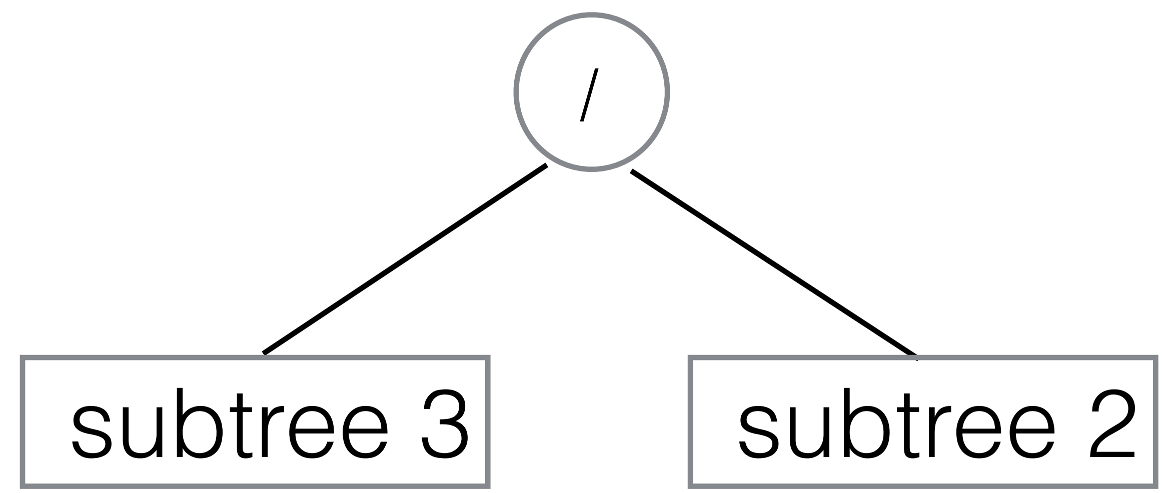
subtree 2

Arithmetic Trees

- **a * (b + c)** / (m - 2)



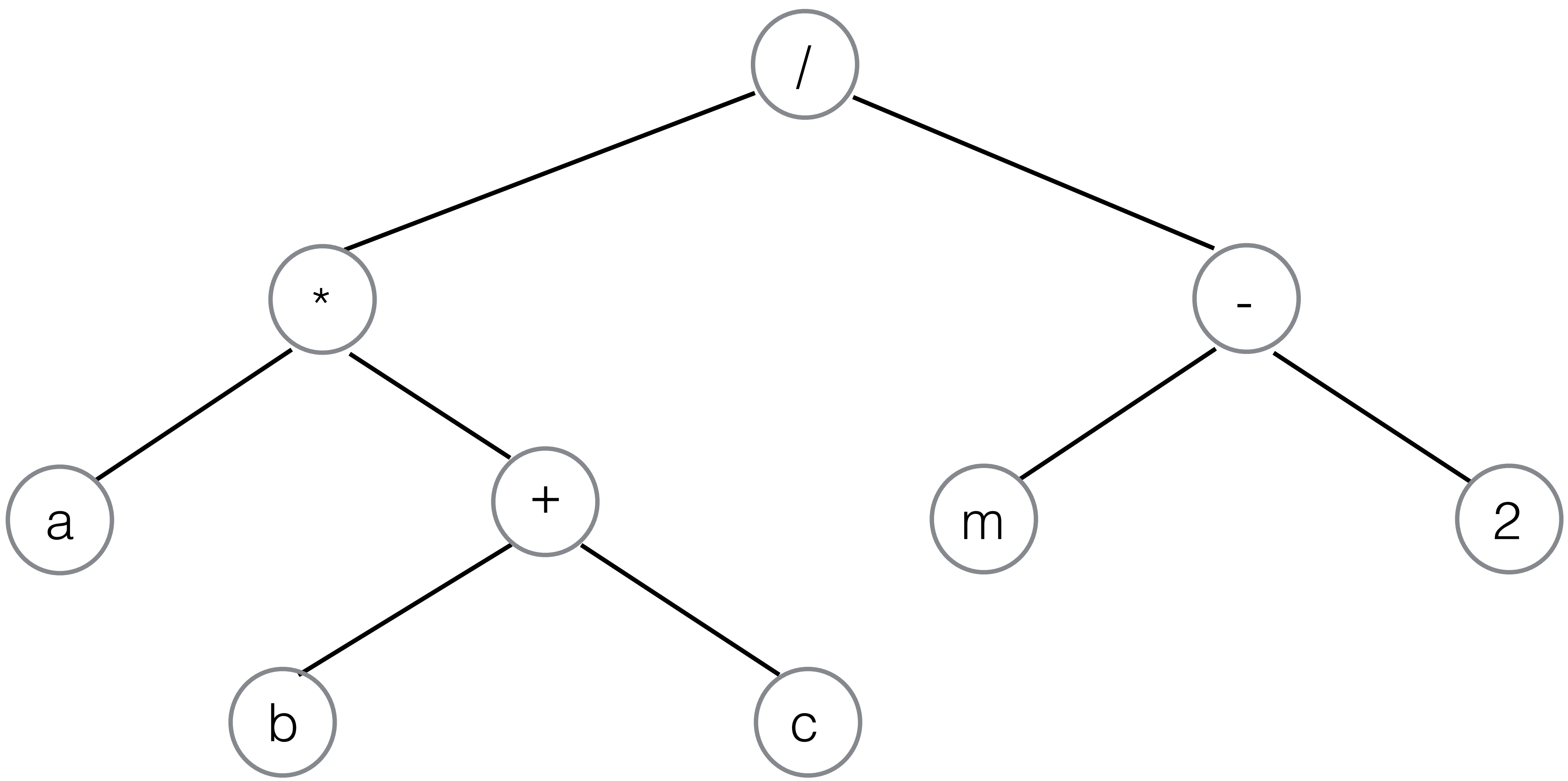
subtree 3



finally

Arithmetic Trees

- **Q.** in-order traversal generates: … ? (that's L,V,R)



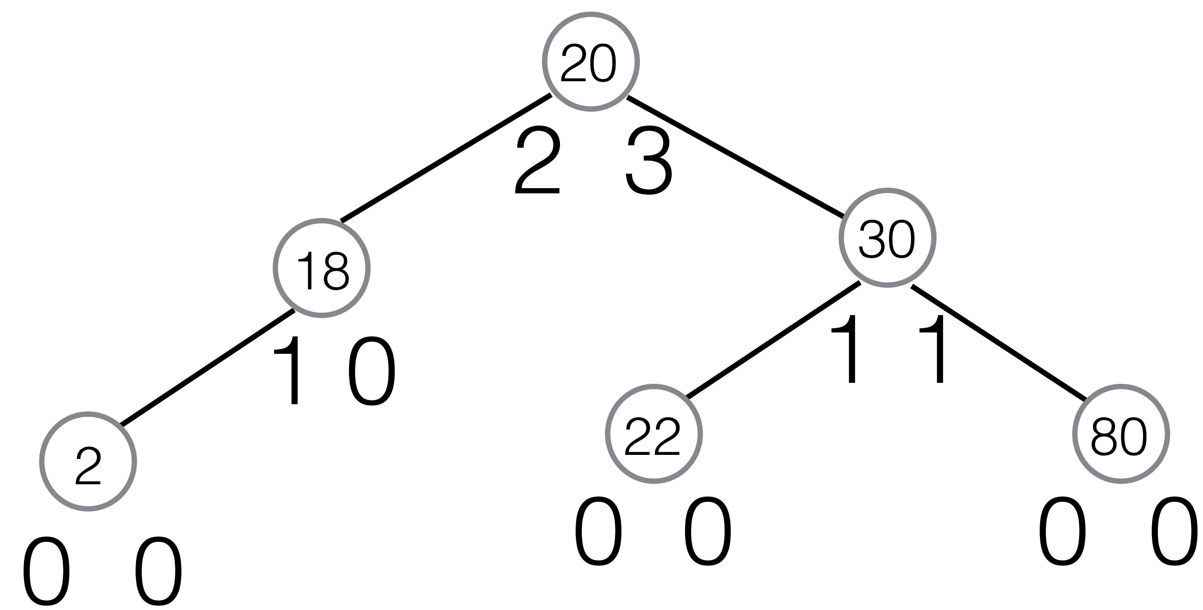
see also: Reverse Polish Notation

Binary Search Tree (BST)

- uses a **binary tree**
- data stored in any node is unique
- any data in **left** subtree is **less than root**
- any data in **right** subtree is **greater than root**
- left and right subtrees are also binary trees

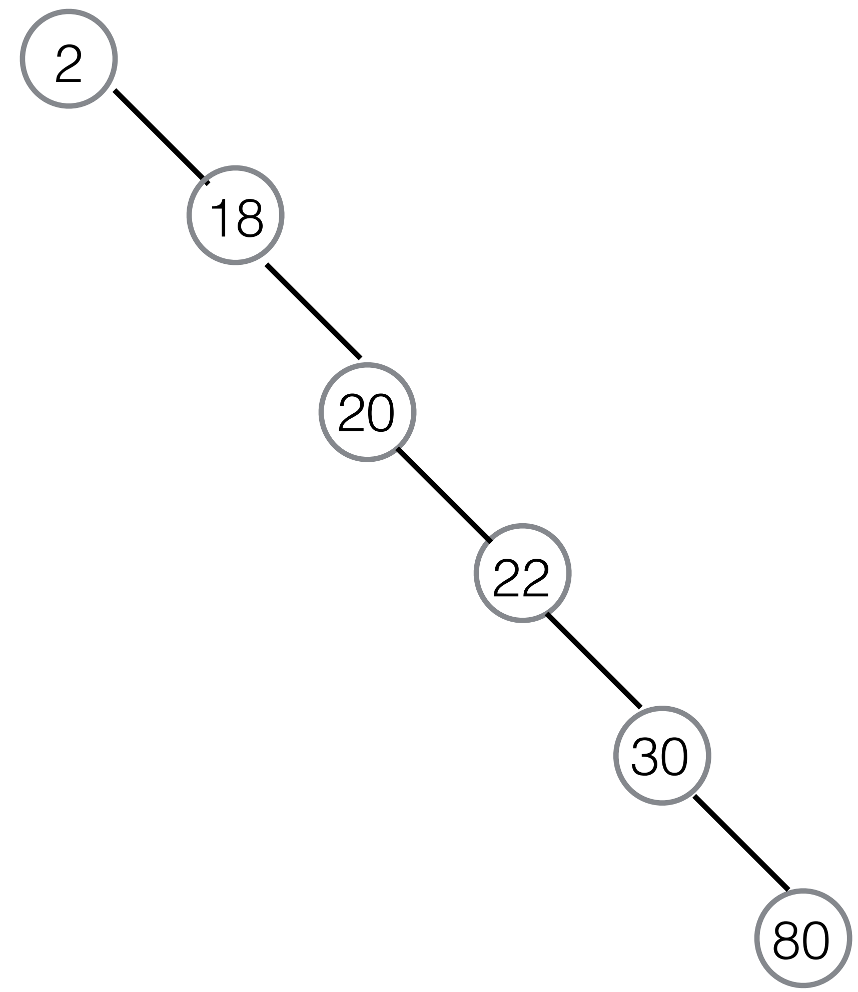
Balance

- A binary tree is perfectly **balanced** if
  - total nodes in left and right trees differs by max 1
  - levels in left and right trees differs by max 1
- To find if 80 is in this tree:
  - compare: 20, 30, 80



Balance

- Not balanced, but still a binary search tree:
  - 6 comparisons needed to find 80
- Search on BST
  - worst case **O(n)**
  - average case **O(log n)**



Balance

- A tree will be balanced if we *insert* values in particular order
  - 20, 18, 30, 2, 22, 80 - first tree
  - 2, 18, 20, 22, 30, 80 - second tree
- If we **sort** the data into a list or array first we can create a perfectly balanced tree:
  - 2, 26, 30, 34, 56, 60, **65**, 70, 80, 94, 96, 98, 99
- *Then* we can choose the **mid value** - 65 - insert that first, split into a left and right list - choose mids of those, and so on.
  - so insertion order will be 65, 30, 2, 26, 56, 34, 60…