CSIT881
Programming and Data Structures

Tree

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Dr. Joseph Tonien

Objectives

- Non-linear data structure: Tree
- Binary Tree
- Binary Search Tree

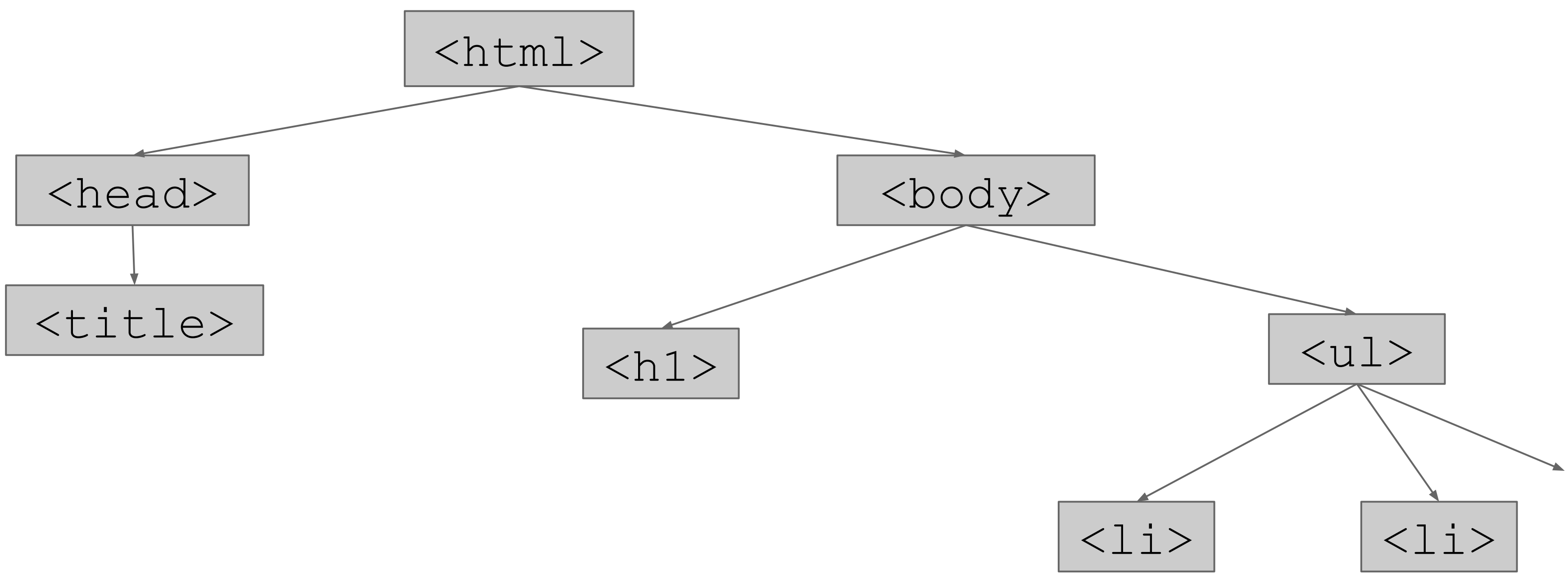
Tree example

A tree can be used to represent the parent-child relationship between elements of a web document.

```
<html>
  <head>
    <title>FIFA</title>
  </head>

  <body>

    <h1>Group A</h1>
    <ul>
      <li>Uruguay</li>
      <li>Russia</li>
      <li>Saudi Arabia</li>
      <li>Egypt</li>
    </ul>

  </body>
</html>
```

Tree example

A tree can be used to represent the parent-child relationship between elements of a web document.



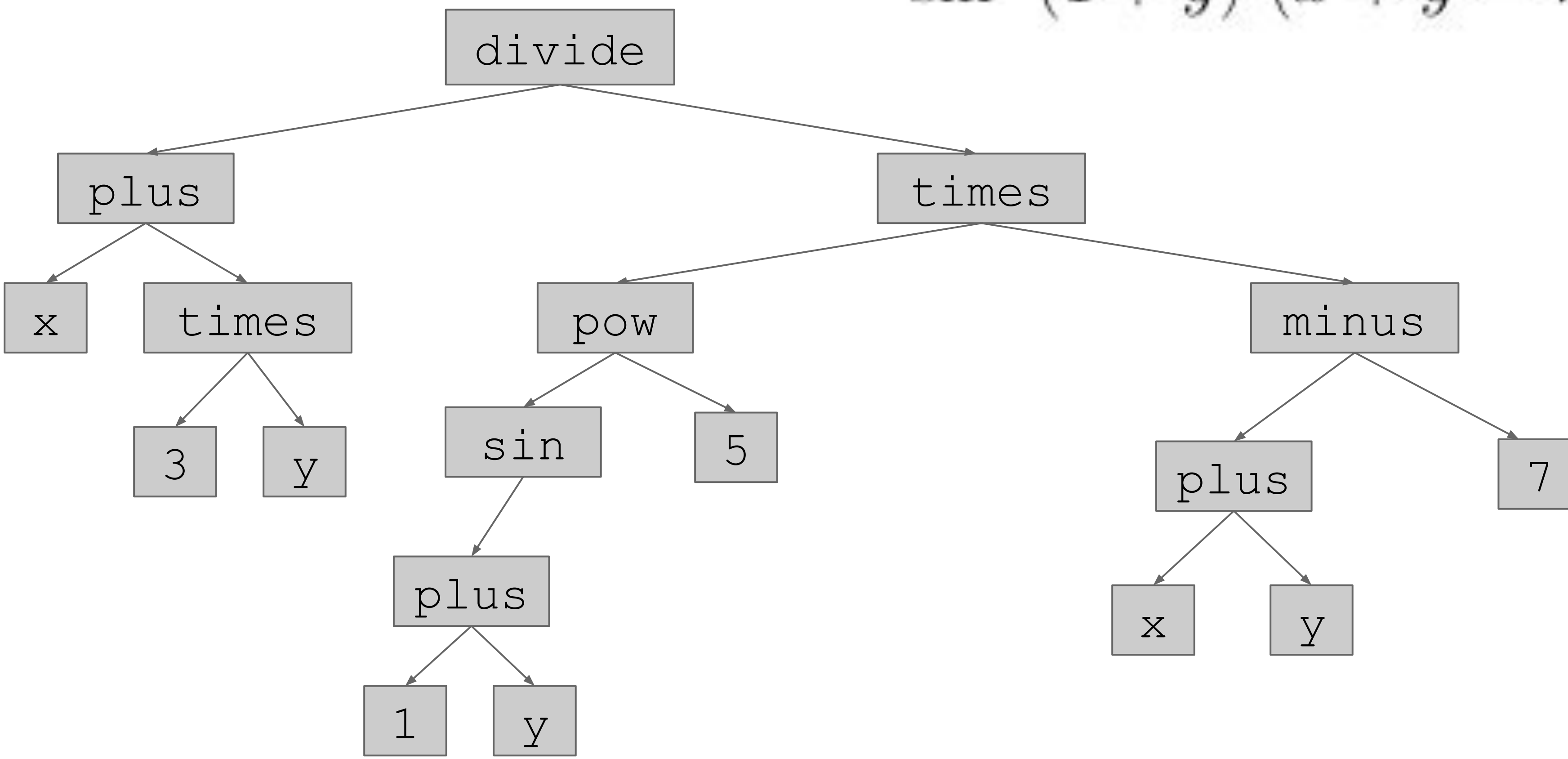
Using tree structure to calculate CSS value via Inheritance Rule:

some CSS properties (such as color and font-family) by default inherit values set on the parent element.

Tree example

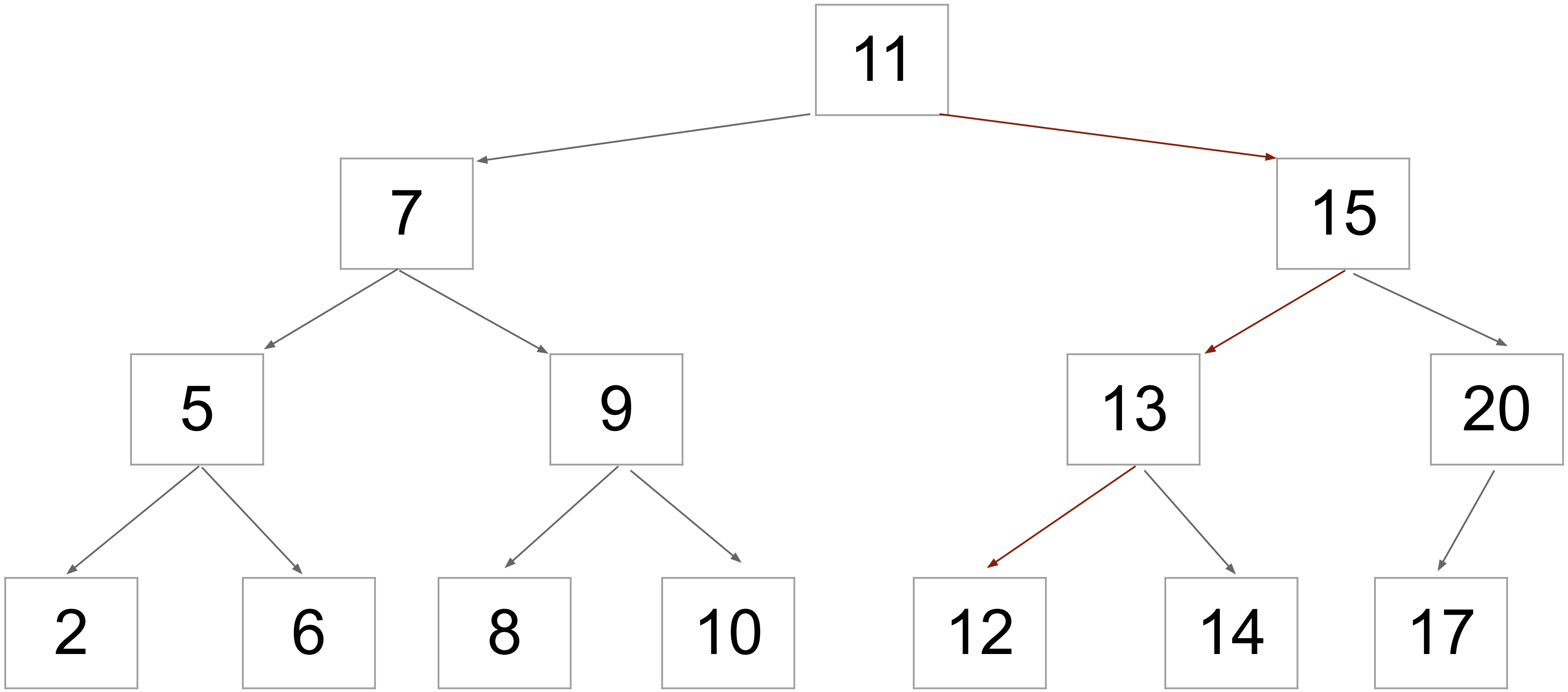
A tree can be used to represent a mathematical expression.

$$\frac{x + 3y}{\sin^5(1+y)\,(x+y-7)}$$



Tree example

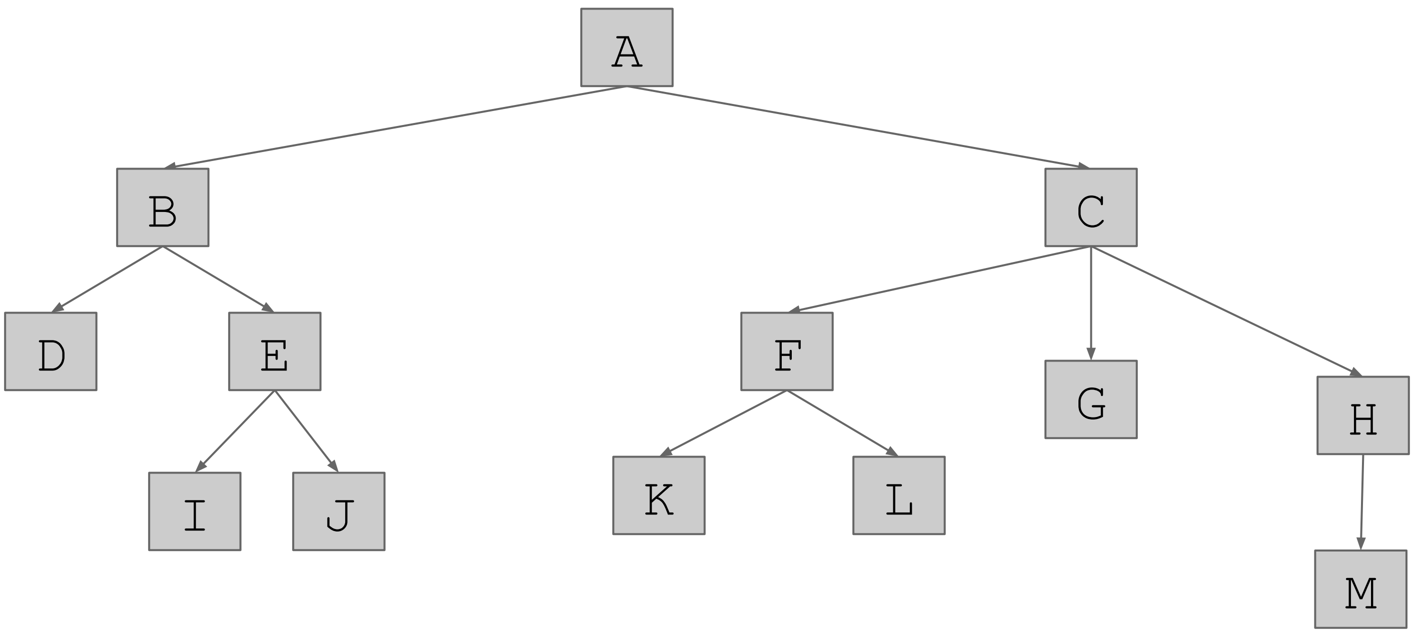
Searching in a binary search tree is very fast since each node stores a key **greater than** all the keys in the node's **left subtree** and **less than** those in its **right subtree**.



searching for 12

Tree terminology

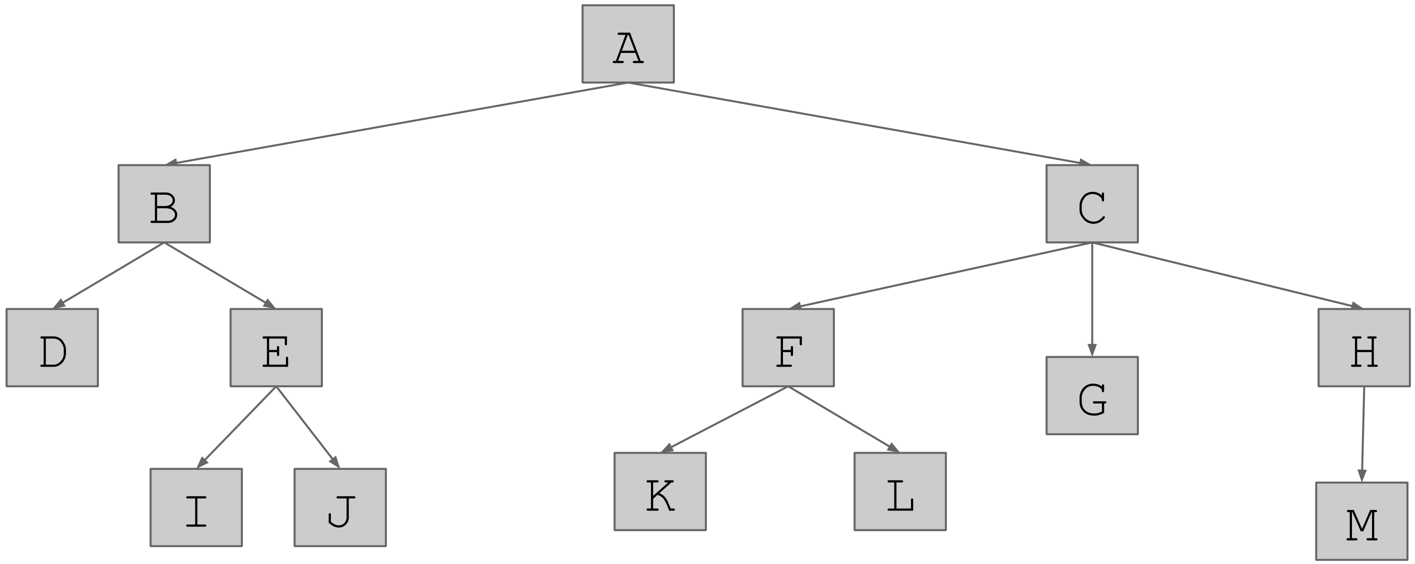
- **root node**: the only node in the tree that has no parent; any other node has a unique parent;
- **leaf node**: a node that has no children;



Root node: A.
Leaf nodes: D, I, J, K, L, G, M.

Tree terminology

- **sibling**: a node that shares the same parent;

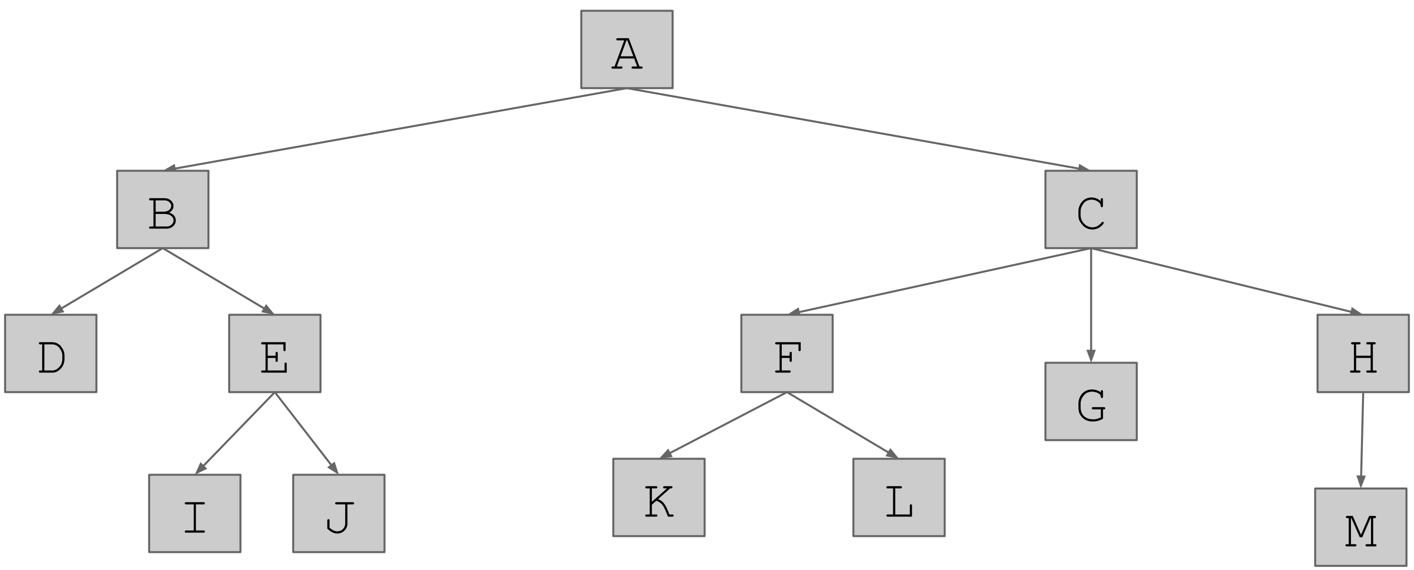


Sibling nodes:

- B, C
- D, E
- I, J
- F, G, H
- K, L

Tree terminology

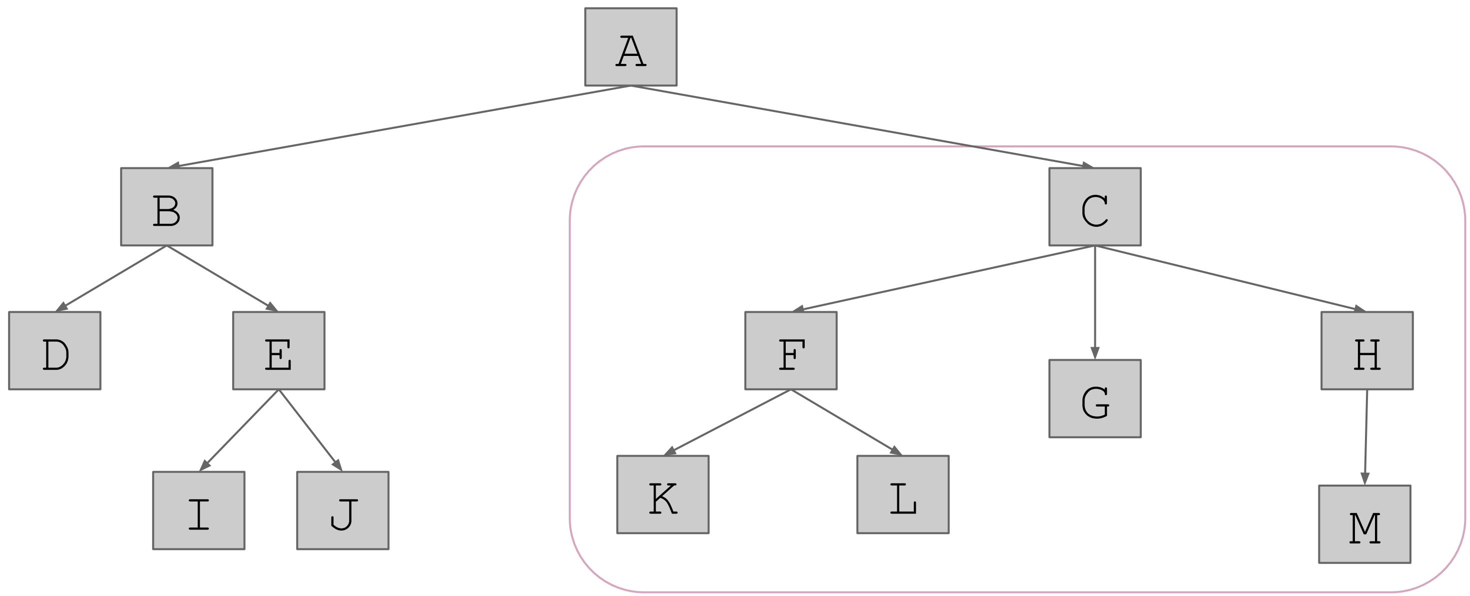
- **ancestor**: a node reachable by repeated travelling from child to parent;
- **descendant**: a node reachable by repeated travelling from parent to child;



Ancestor of E: B, A
Descendant of C: F, G, H, K, L, M

Tree terminology

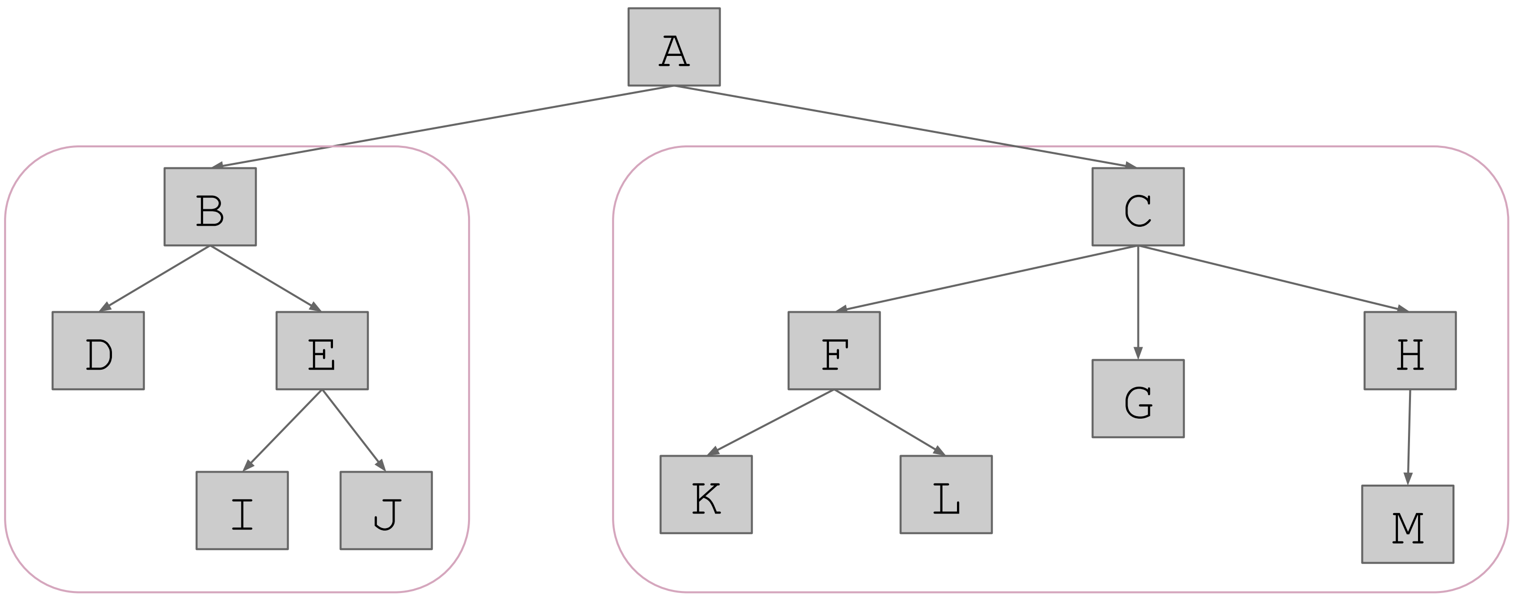
- **subtree** rooted at a node: a tree consisting of the node together with all its descendants;



Subtree **rooted** at node C

Tree terminology

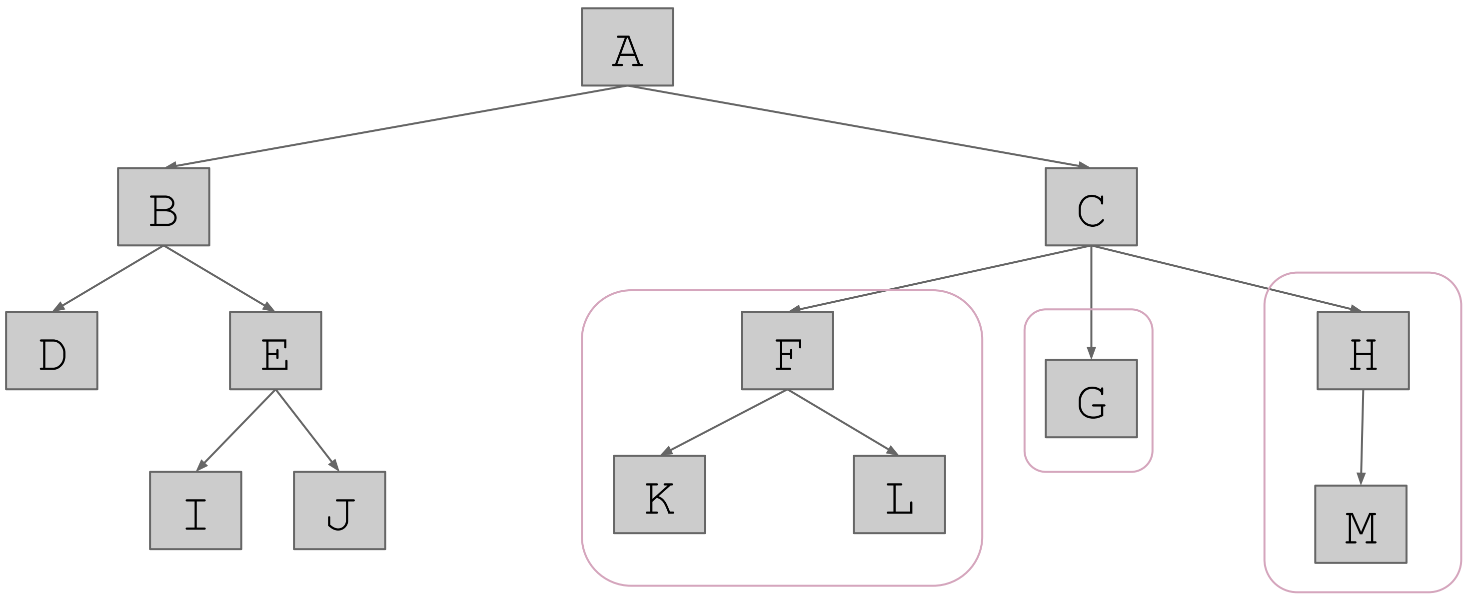
- **subtree** of a node: a subtree rooted at the node’s children;



Node A has two subtrees: left subtree rooted at B and right subtree rooted at C

Tree terminology

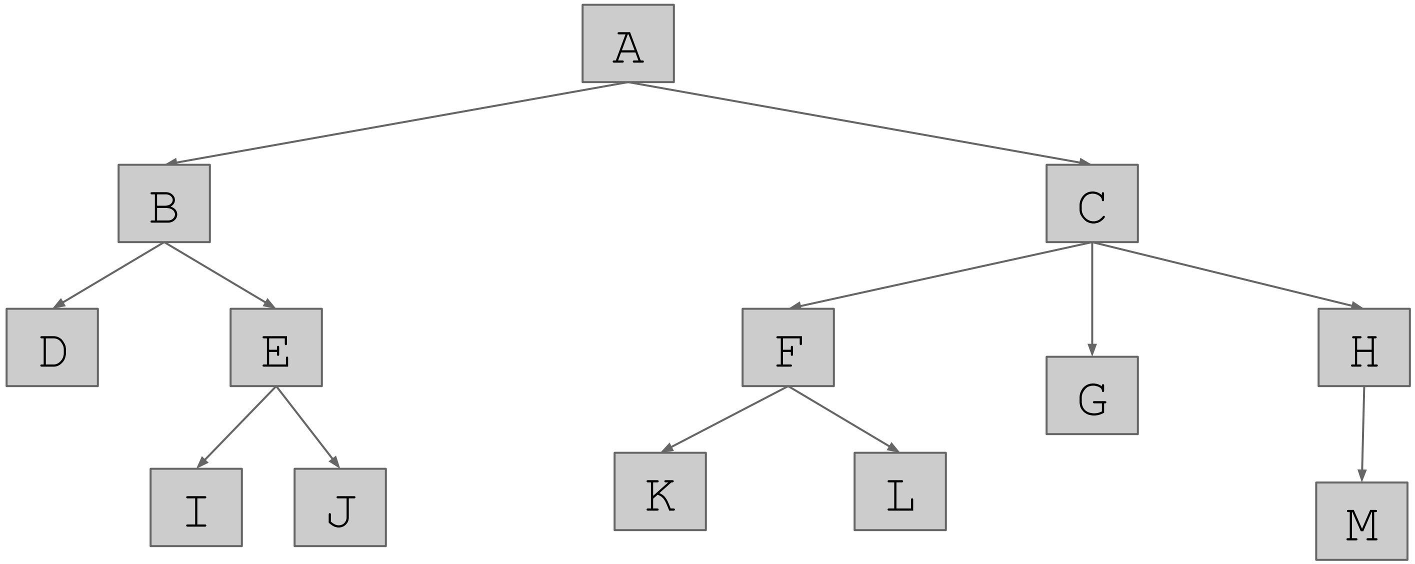
- **subtree** of a node: a subtree rooted at the node's children;



Node C has three subtrees: left subtree, middle subtree and right subtree

Tree terminology

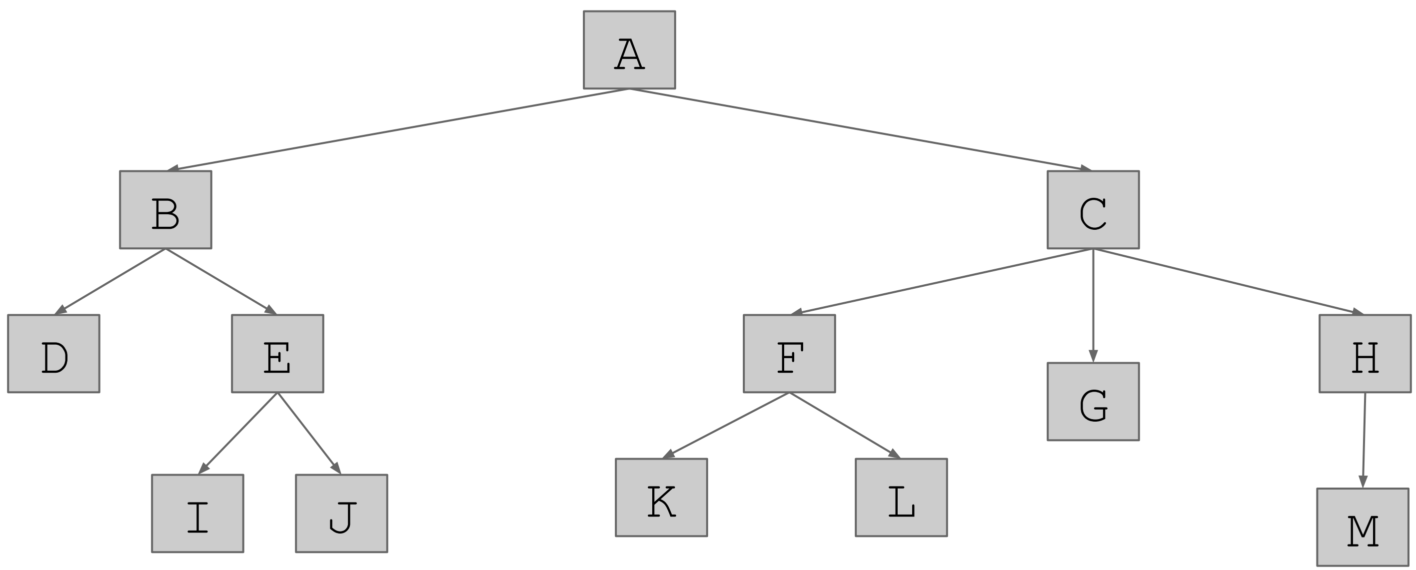
- **degree of a node**: the number of its children;



Nodes of degree 0: D, I, J, K, L, G, M
Nodes of degree 1: H
Nodes of degree 2: A, B, E, F
Nodes of degree 3: C

Tree terminology

- **degree of a tree**: each node has a degree; the degree of the tree is the maximum of all node degrees;

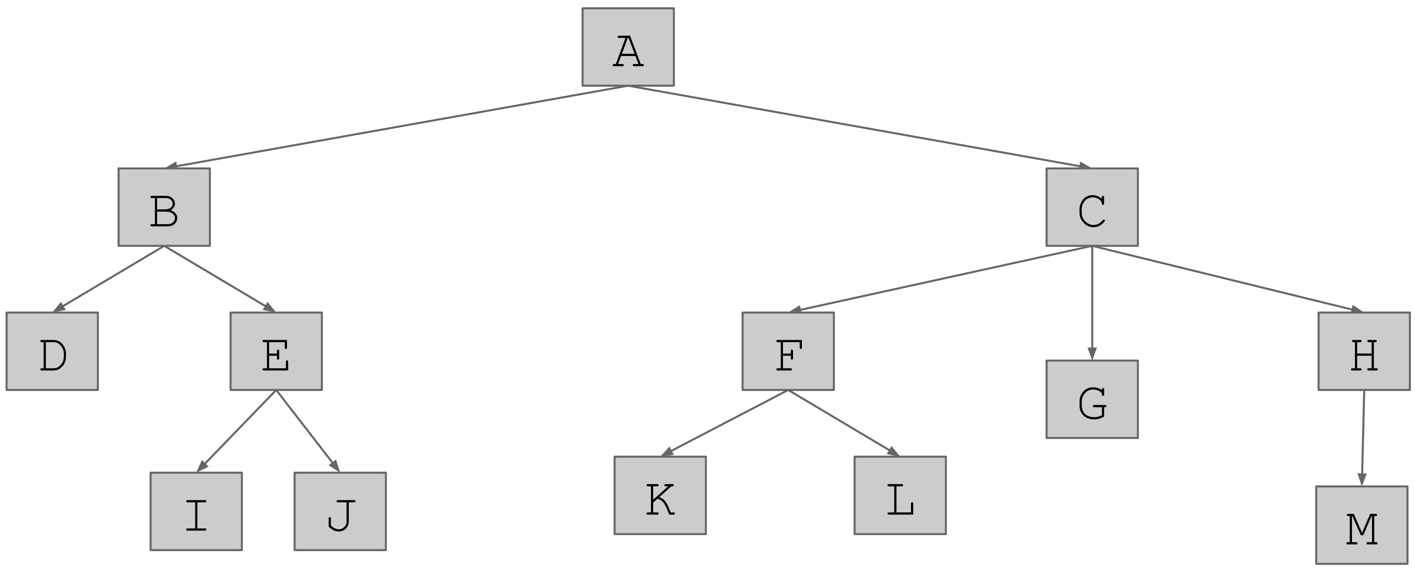


Tree degree = 3

Nodes of degree 0: D, I, J, K, L, G, M
Nodes of degree 1: H
Nodes of degree 2: A, B, E, F
Nodes of degree 3: C

Tree terminology

- **distance** between two nodes: the number of edges along the shortest path connecting the two nodes;

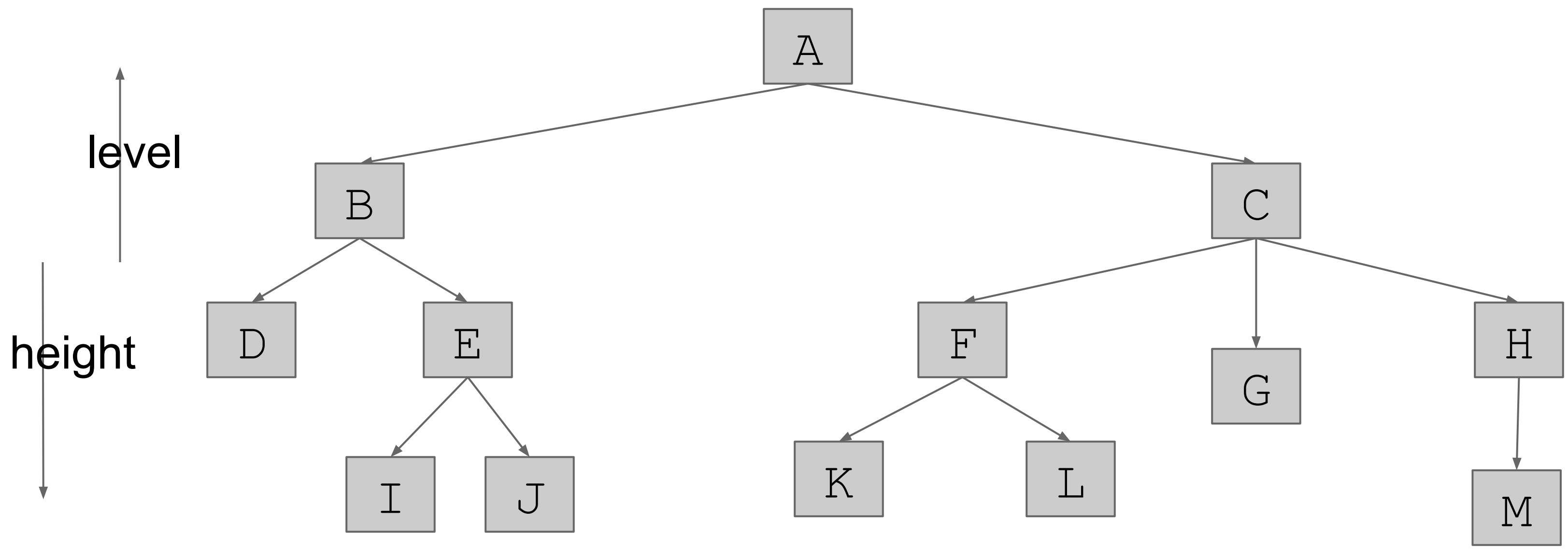


Distance between B and J: 2, shortest path: B -> E -> J

Distance between B and L: 4, shortest path: B -> A -> C -> F -> L

Tree terminology

- **level of a node**: the number of edges along the unique path connecting the node and the root;
- **height of a node**: the number of edges that connect the node to its farthest descendant;



A: level 0

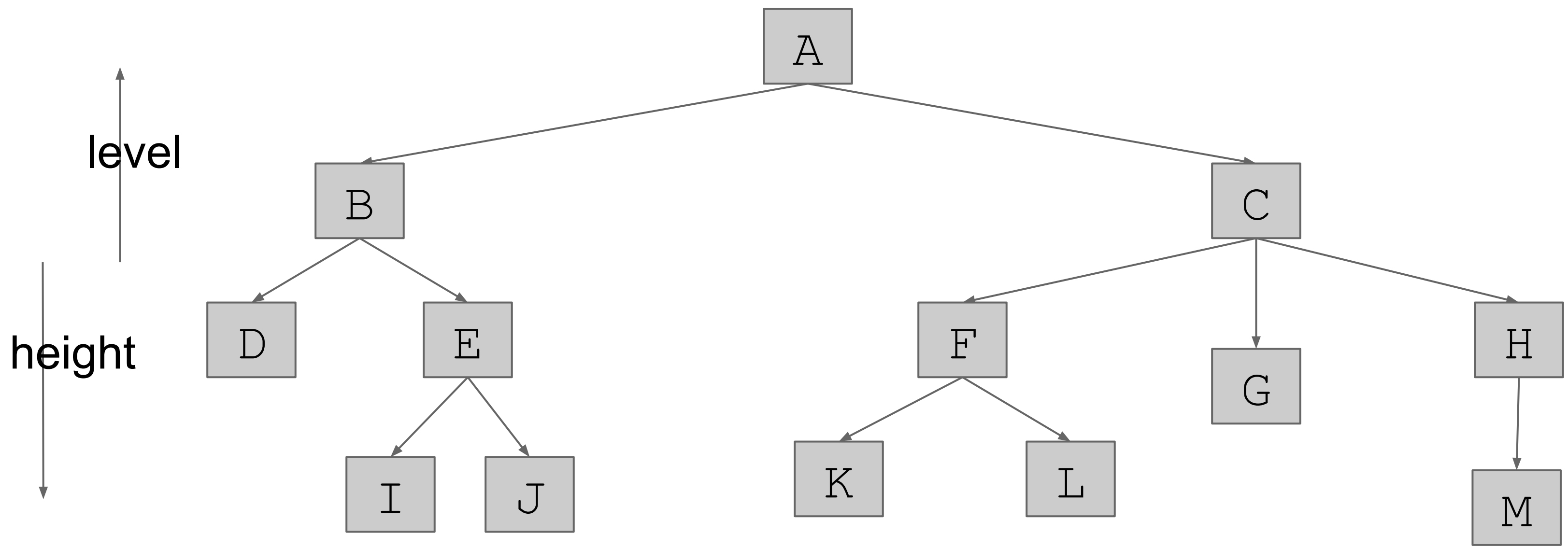
B: level 1
C: level 1

D: level 2
E: level 2
F: level 2
G: level 2
H: level 2

I: level 3
J: level 3
K: level 3
L: level 3
M: level 3

Tree terminology

- **level of a node**: the number of edges along the unique path connecting the node and the root;
- **height of a node**: the number of edges that connect the node to its farthest descendant;



A: level 0, height 3

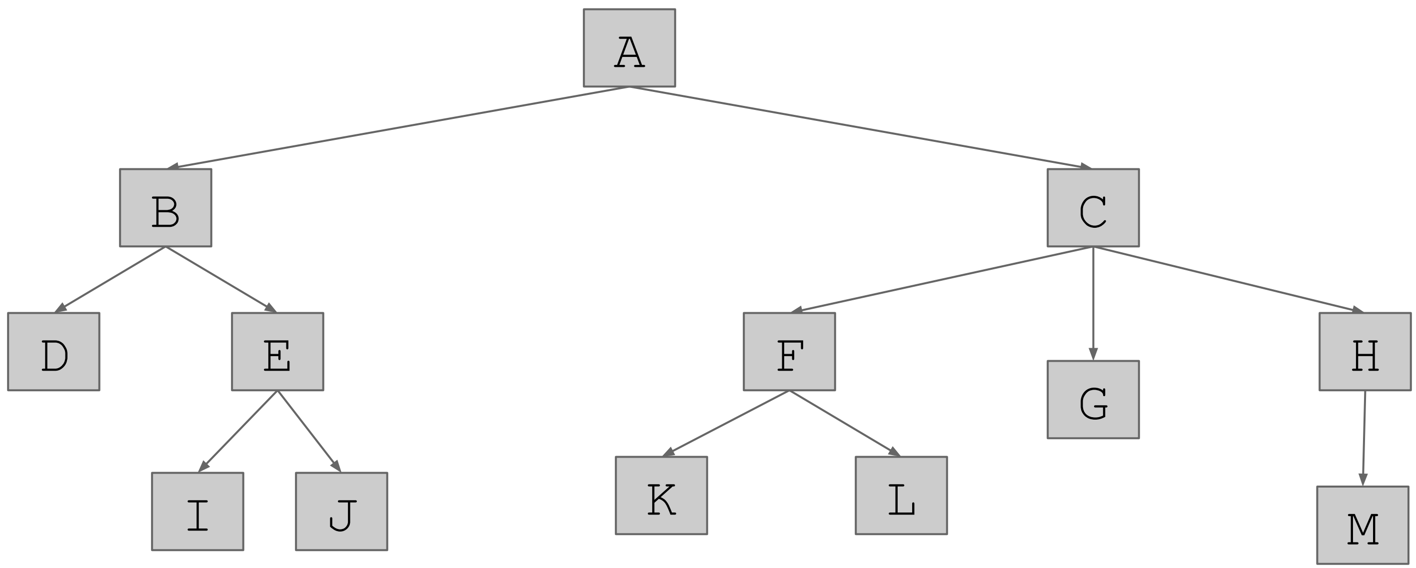
B: level 1, height 2
C: level 1, height 2

D: level 2, height 0
E: level 2, height 1
F: level 2, height 1
G: level 2, height 0
H: level 2, height 1

I: level 3, height 0
J: level 3, height 0
K: level 3, height 0
L: level 3, height 0
M: level 3, height 0

Tree terminology

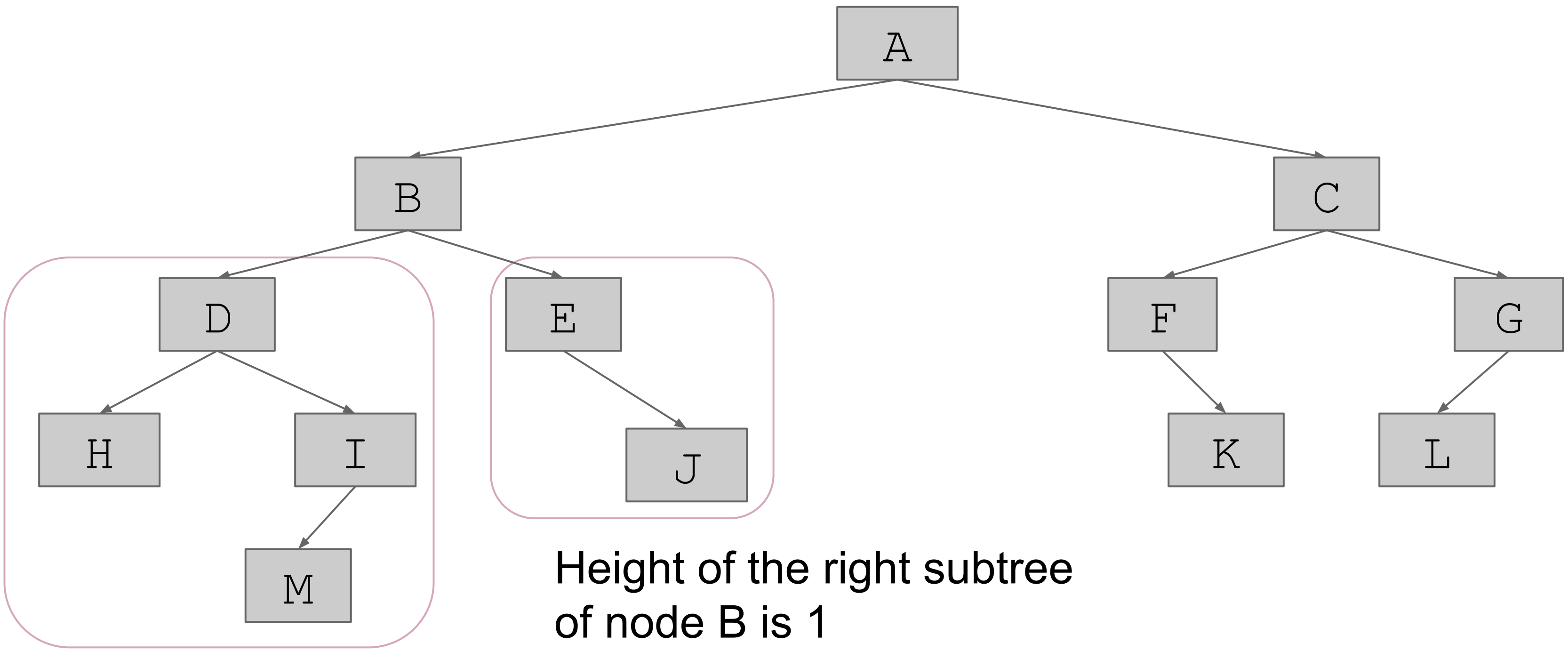
- **height of a tree**: the number of edges that connect the root to its farthest leaf



Height of the tree: 3

Tree terminology

- **height of a subtree**: the number of edges that connect the root of the subtree to its farthest leaf



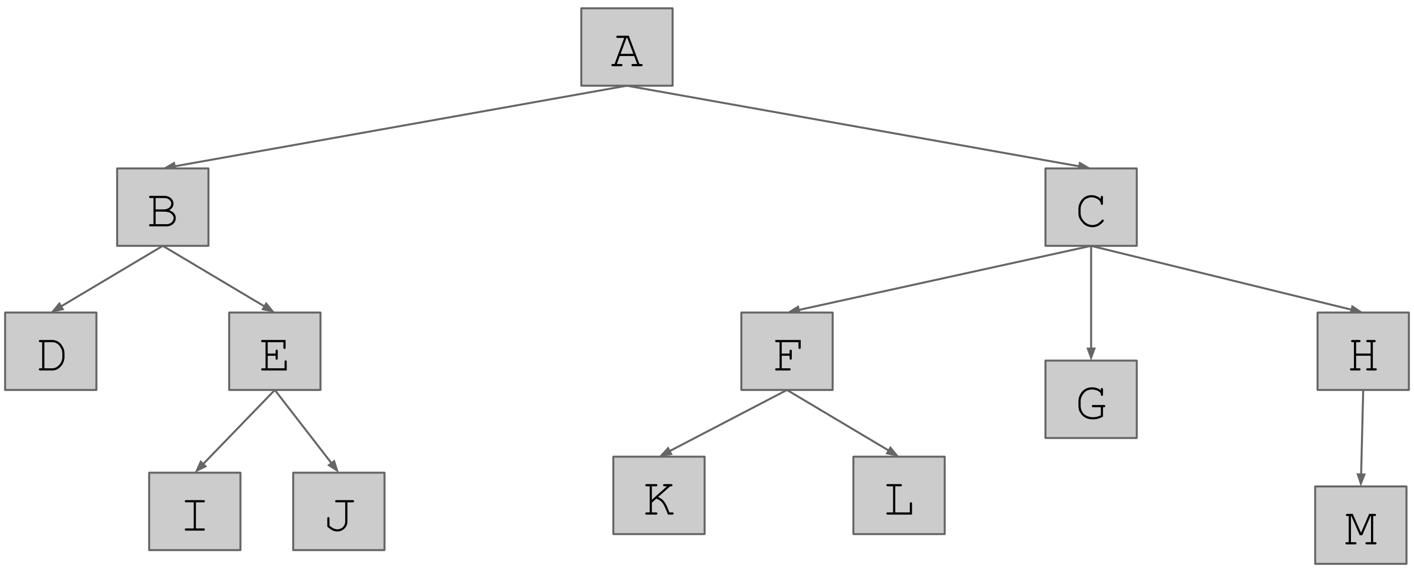
Height of the right subtree of node B is 1

Height of the left subtree of node B is 2

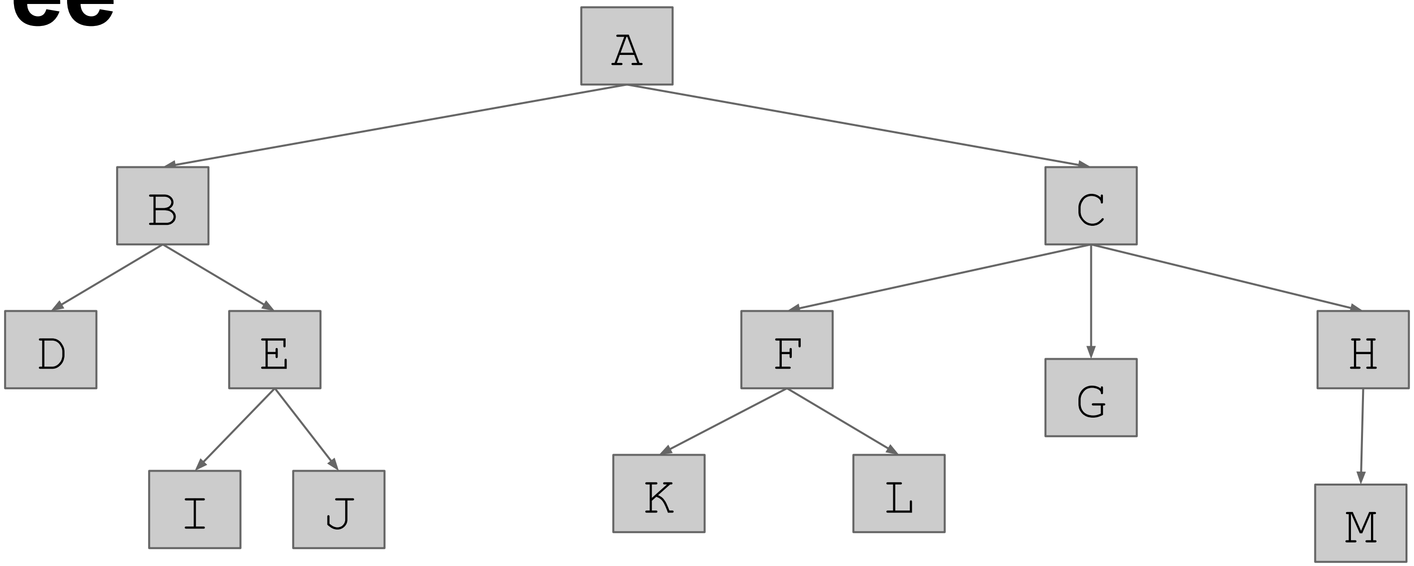
Tree

A tree is an abstract data structure:

- has at most one root node;
- each node has a number of children nodes.



Tree

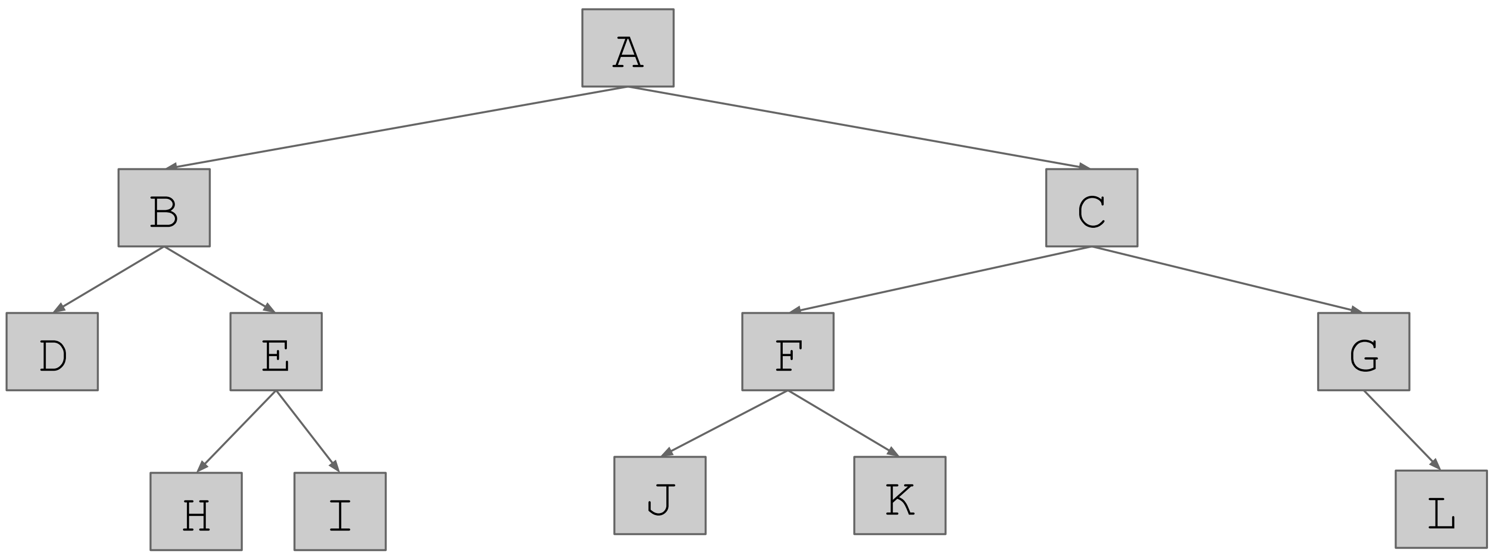


Some common operations on tree:

- Add a node
- Remove a node
- Search for a node
- Travel around a tree

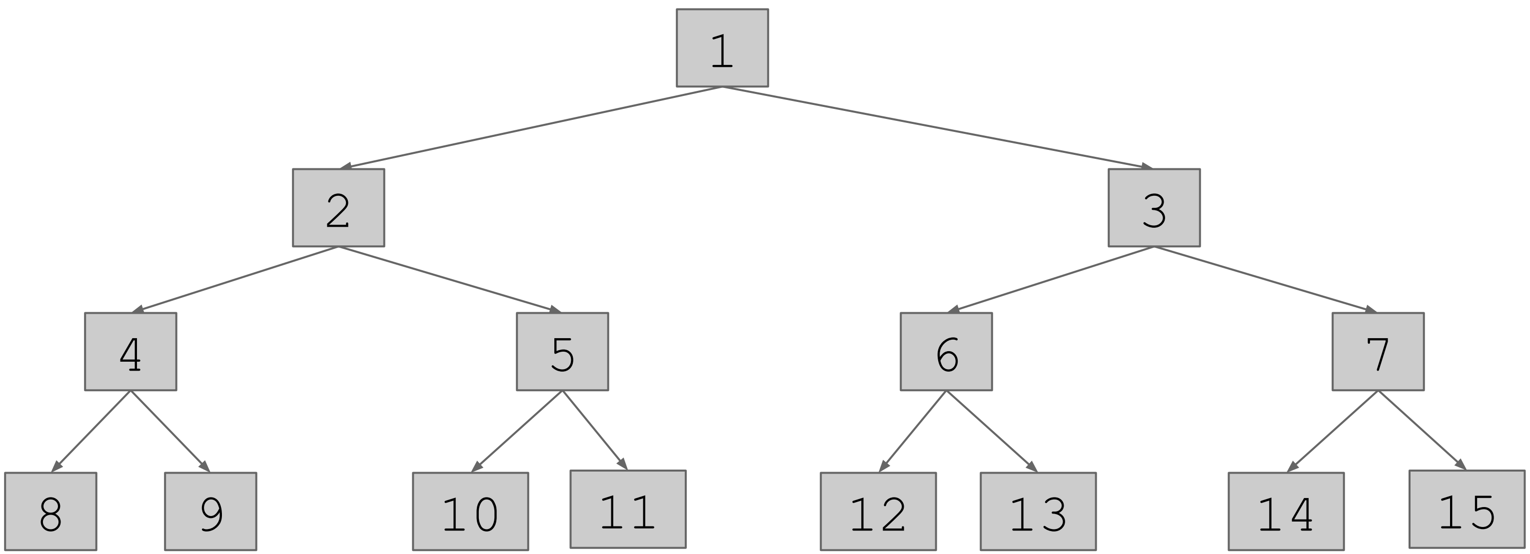
Binary Tree

Binary tree: each node has at most two children



Binary Tree

A perfect binary tree of height h has $n = 2^{h+1}-1$ number of nodes, so $h \approx \lg(n)$

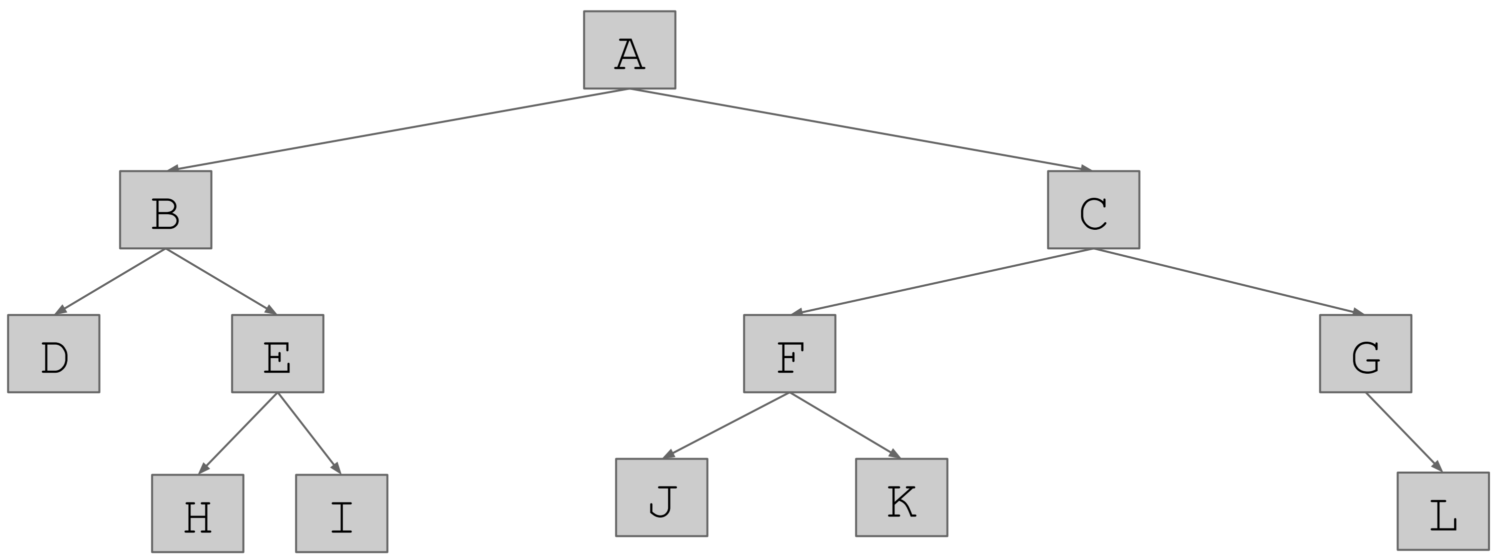


A perfect binary tree of height $h=3$ has $n=2^4-1=15$ nodes

Binary Tree

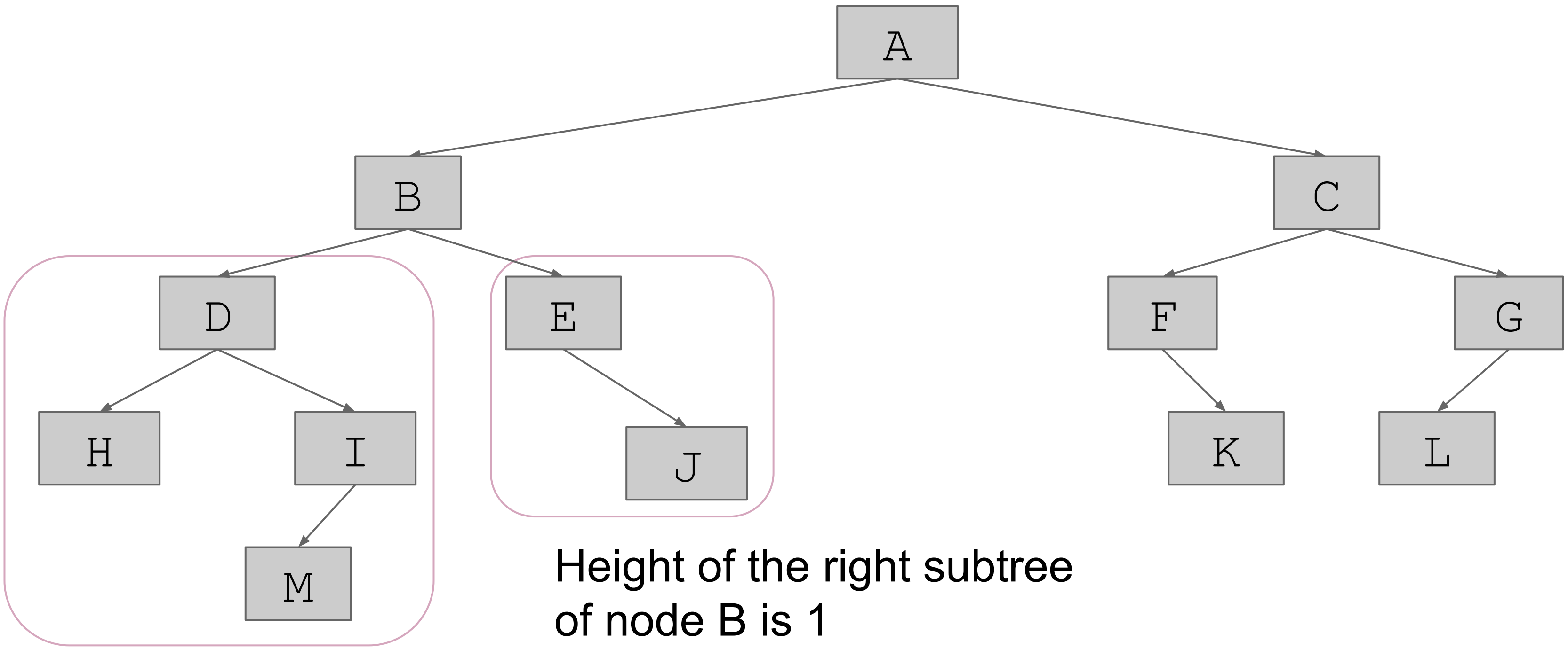
In a random binary tree, the height $h = O(\lg(n))$

So if an algorithm requires a constant number of traveling up and down the tree then it is very fast as the running time is $O(h) = O(\lg(n))$



Binary Tree

- **balance factor of a node** = `height of its right subtree - height of its left subtree`



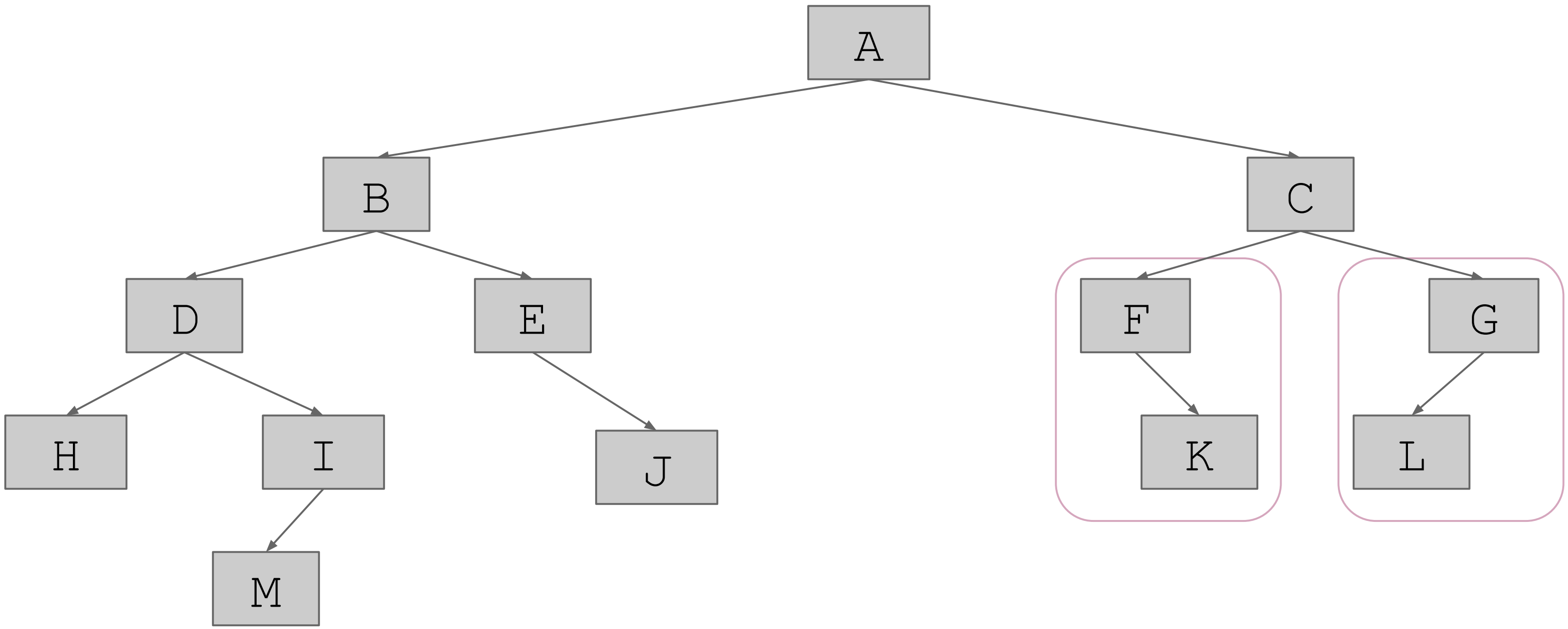
Height of the right subtree of node B is 1

Height of the left subtree of node B is 2

`Balance factor of node B = -1`

Binary Tree

- **balance factor of a node** = `height of its right subtree - height of its left subtree`

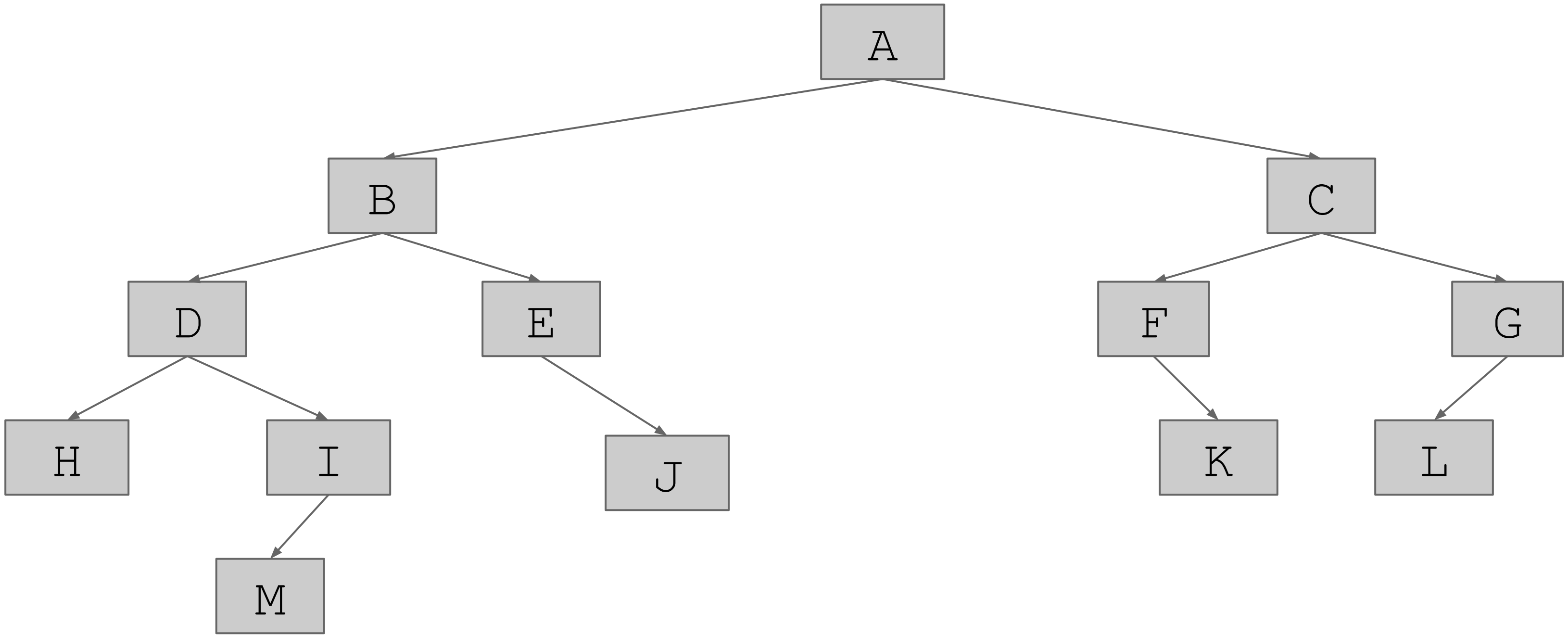


Balance factor of node C = 0

Binary Tree

- If **balance factor of a node** $= 0$: the node is called balanced
- If **balance factor of a node** > 0: the node is called right-heavy
- If **balance factor of a node** < 0: the node is called left-heavy

Binary Tree



Balanced node: C, H, J, K, L, M

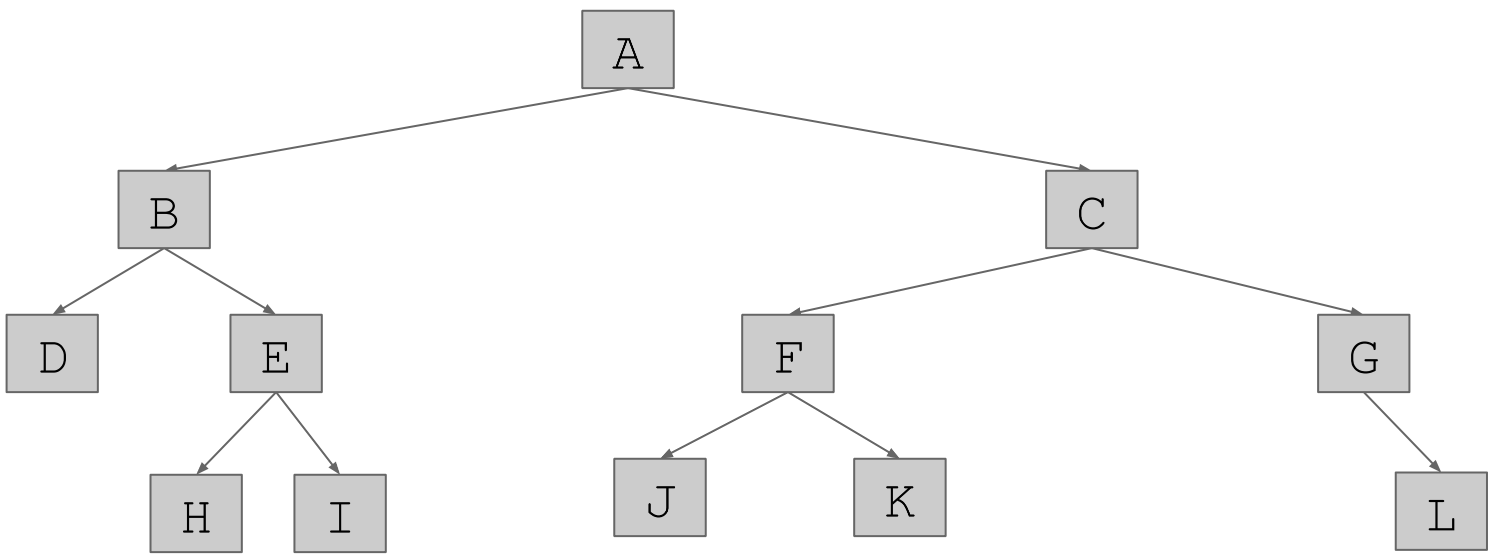
Right-heavy node: D, E, F

Left-heavy node: A, B, G, I

Binary Tree

Tree traversal: a process of iterating the tree - visiting each node once. There are 3 common tree traversals:

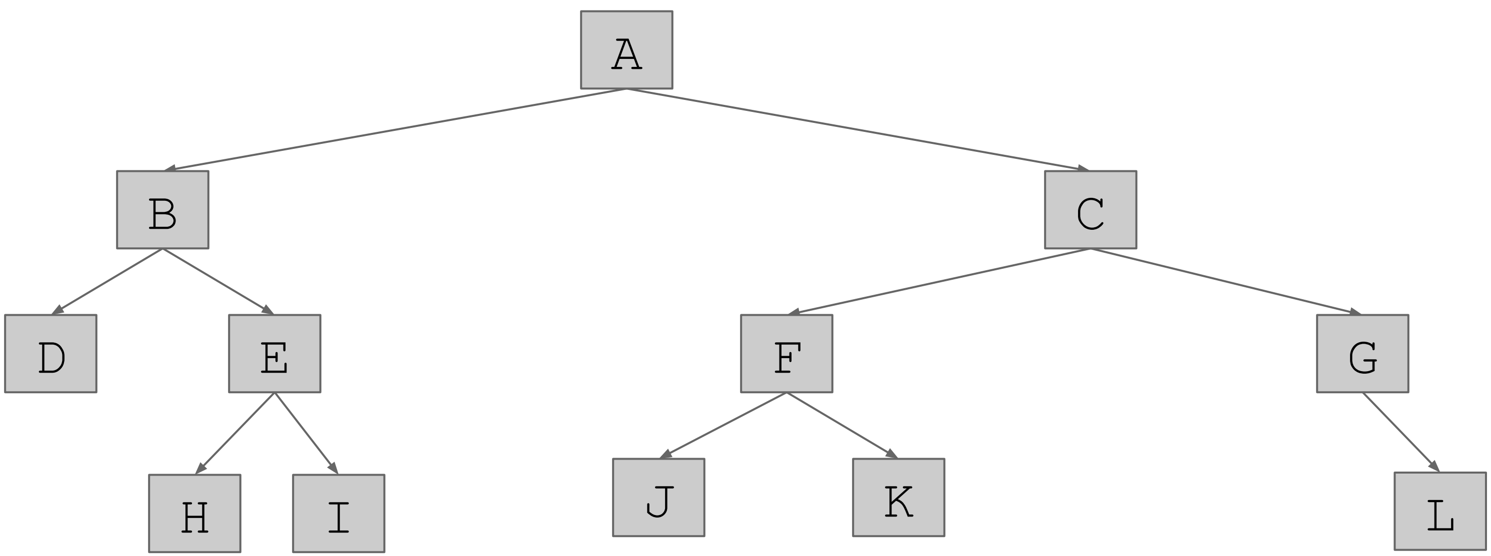
- In-order traversal
- Pre-order traversal
- Post-order traversal



Binary Tree

In-order tree traversal:

- Recursively traverse the current node's left subtree;
- Visit the current node;
- Recursively traverse the current node's right subtree.

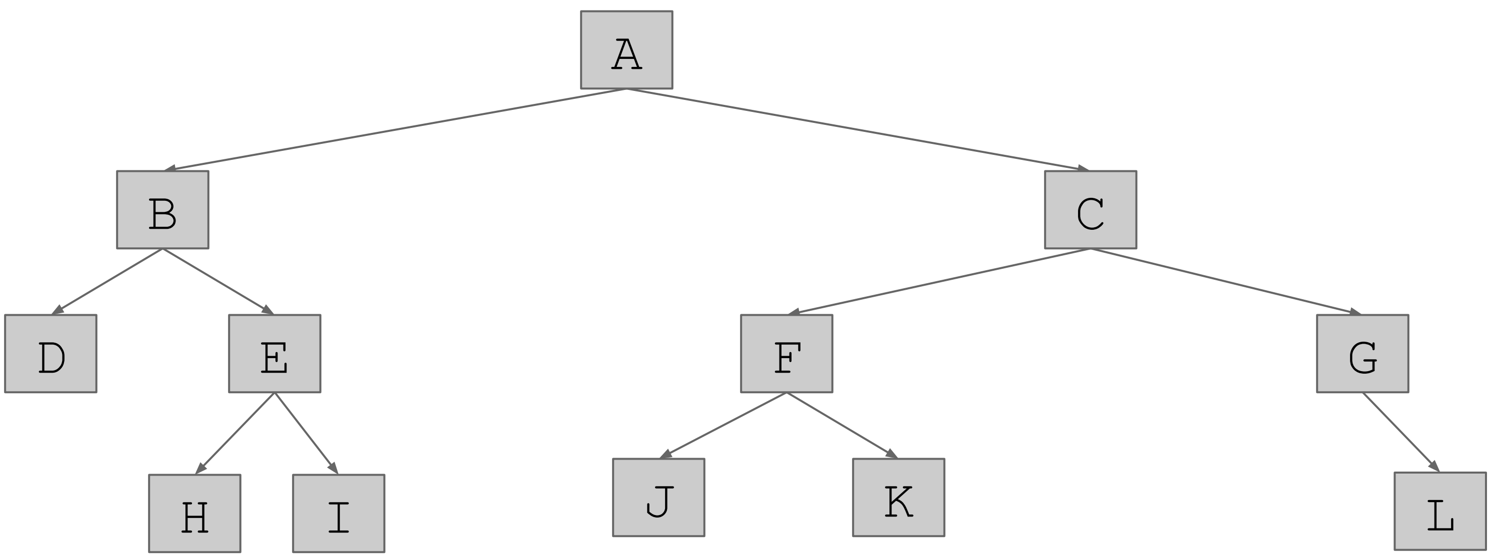


In-order traversal: D, B, H, E, I, A, J, F, K, C, G, L

Binary Tree

Pre-order tree traversal:

- Visit the current node;
- Recursively traverse the current node's left subtree;
- Recursively traverse the current node's right subtree.

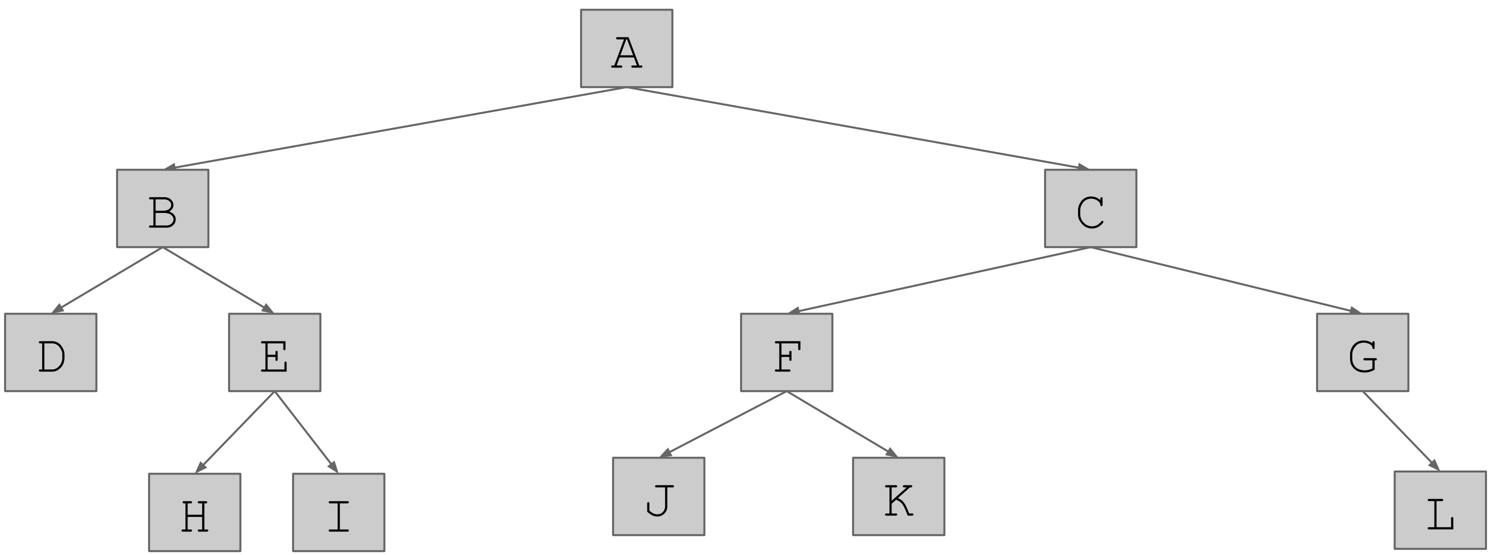


Pre-order traversal: A, B, D, E, H, I, C, F, J, K, G, L

Binary Tree

Post-order tree traversal:

- Recursively traverse the current node’s left subtree;
- Recursively traverse the current node’s right subtree;
- Visit the current node.

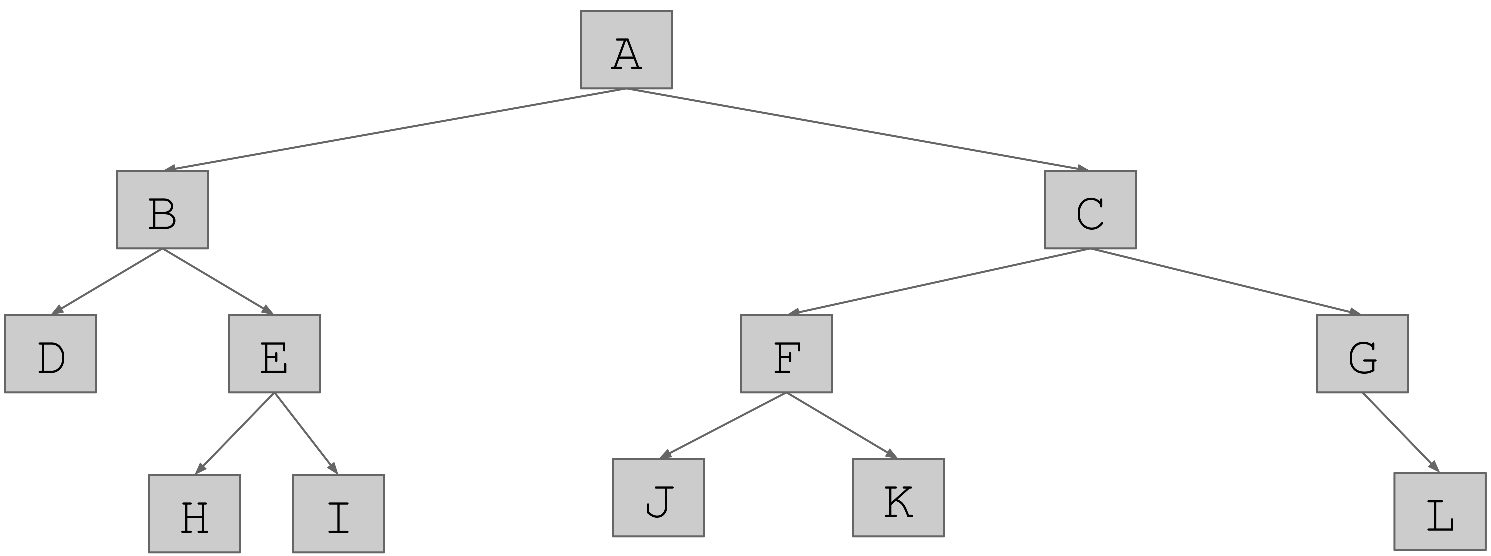


Post-order traversal: D, H, I, E, B, J, K, F, L, G, C, A

Binary Tree

Level-order tree traversal (less common)

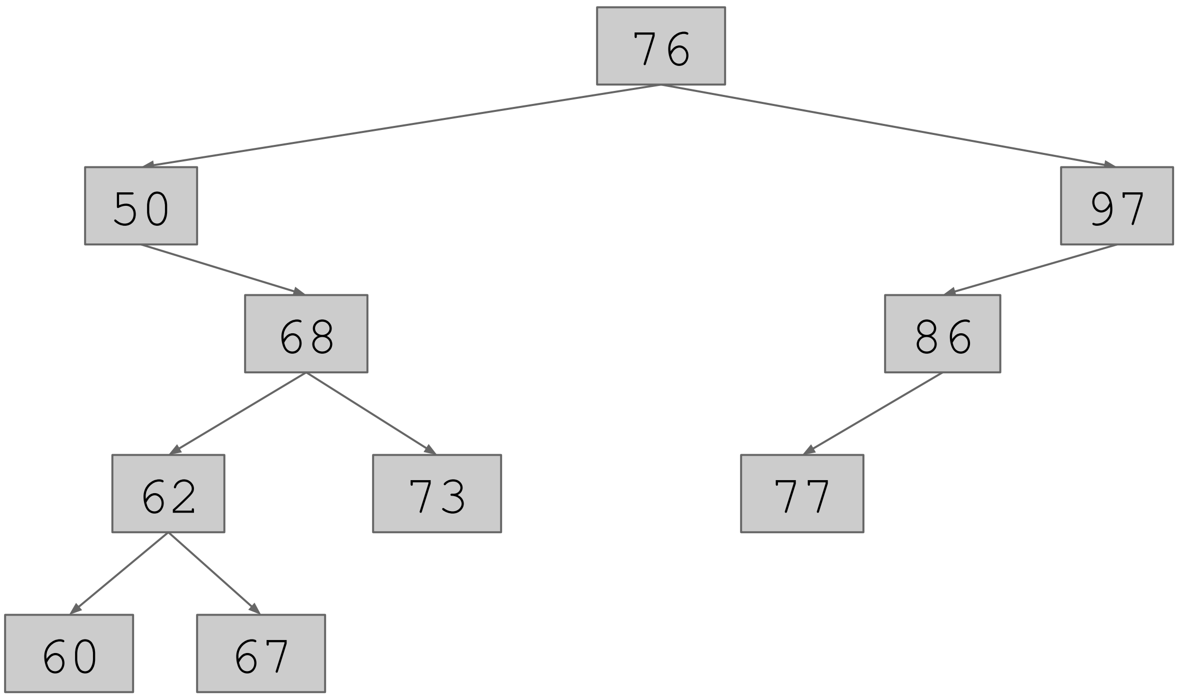
- start from the root
- go level by level
- from left to right



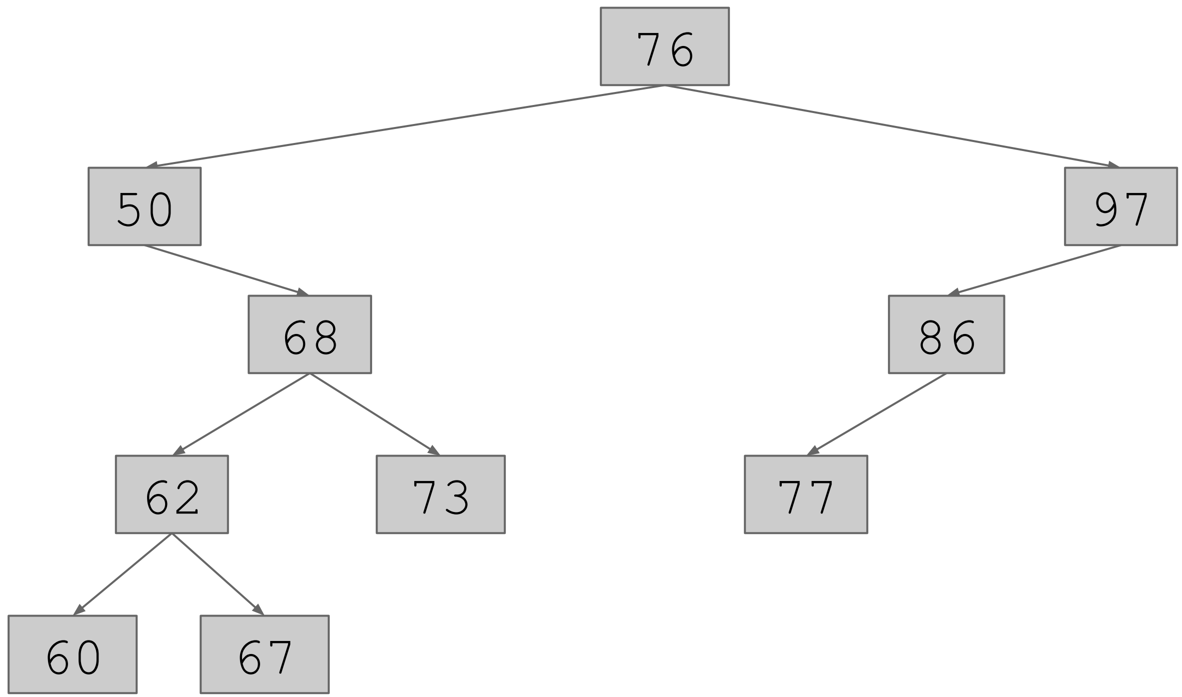
level-order traversal: A, B, C, D, E, F, G, H, I, J, K, L

Binary Search Tree

Binary search tree: a binary tree where each node stores a key **greater than** all the keys in the node's **left subtree** and **less than** those in its **right subtree**



Binary Search Tree



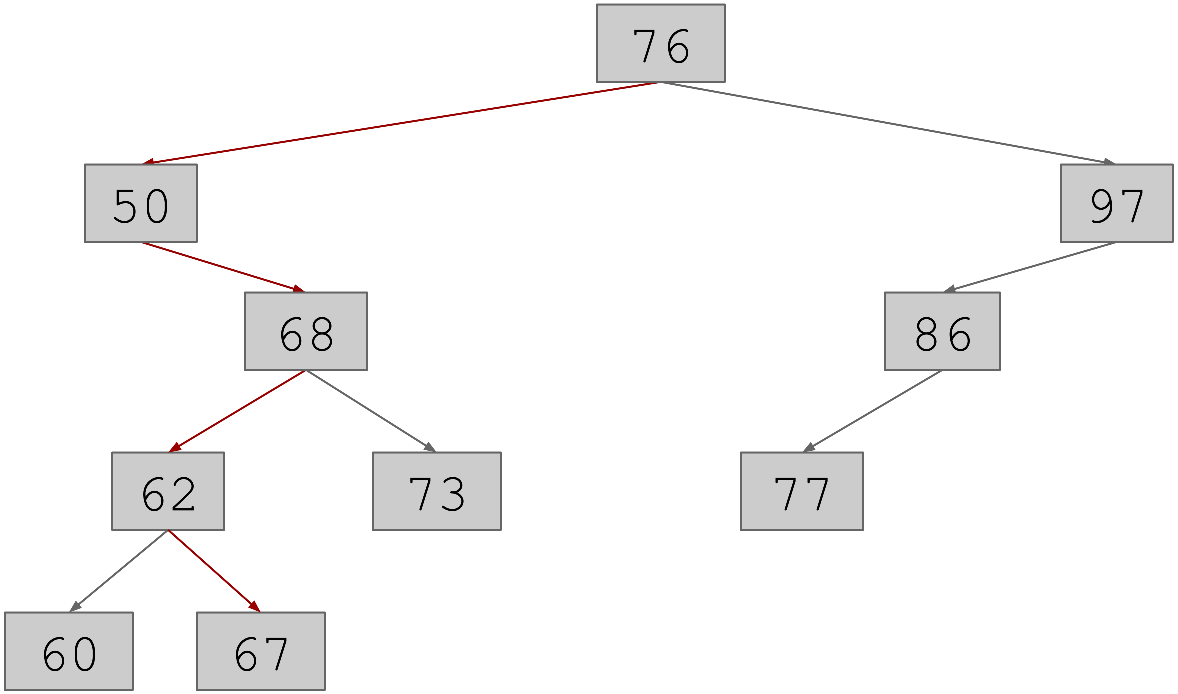
As an example, we usually picture each node of the tree holding an integer. However, in general, each node can hold any data object associated with a key:

- If the data is an integer then the key is the integer itself;
- If the data is a Student object and we want to order by the student number then the key is the student number of that Student object;
- If the data is an Employee object and we want to order by the employee name then the key can be the pair (last name, first name)

Binary Search Tree

Searching in a binary search tree is very fast

Searching for 67:

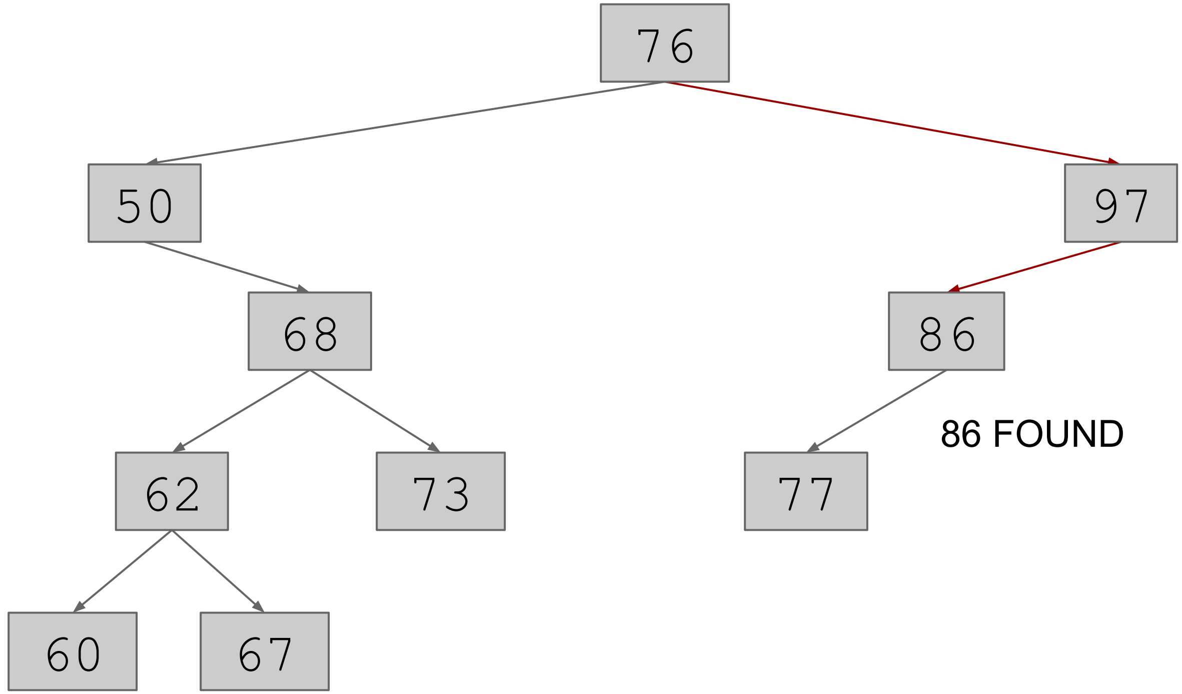


67 FOUND

Binary Search Tree

Searching in a binary search tree is very fast

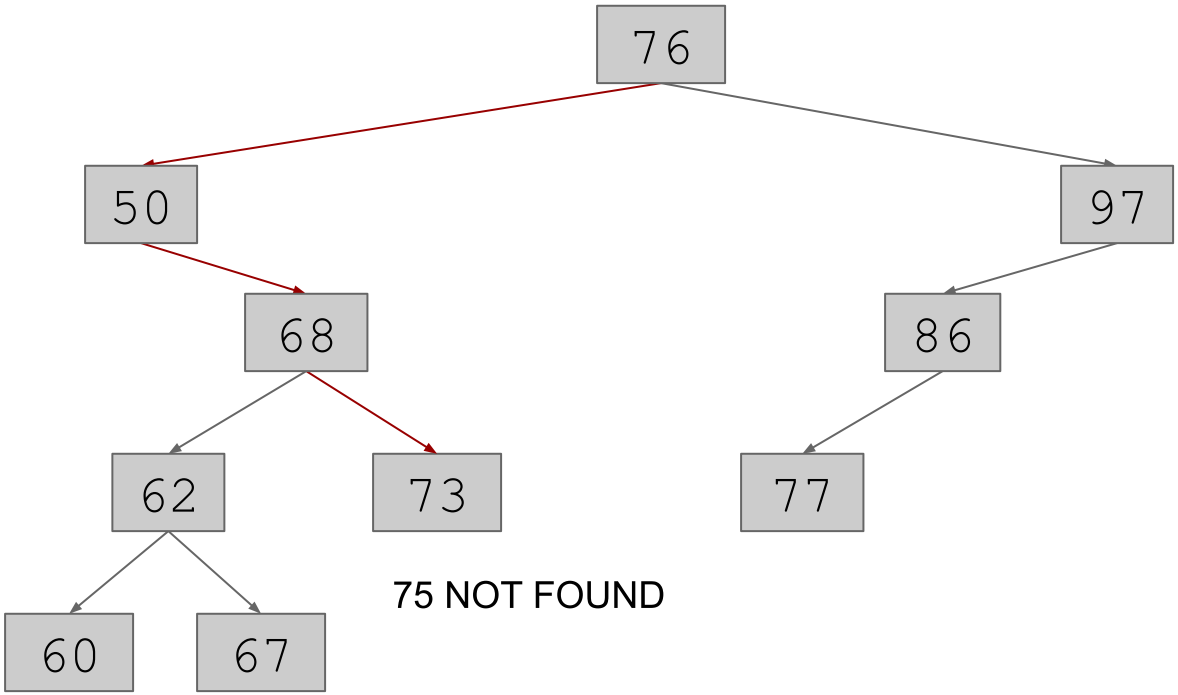
Searching for 86:



Binary Search Tree

Searching in a binary search tree is very fast

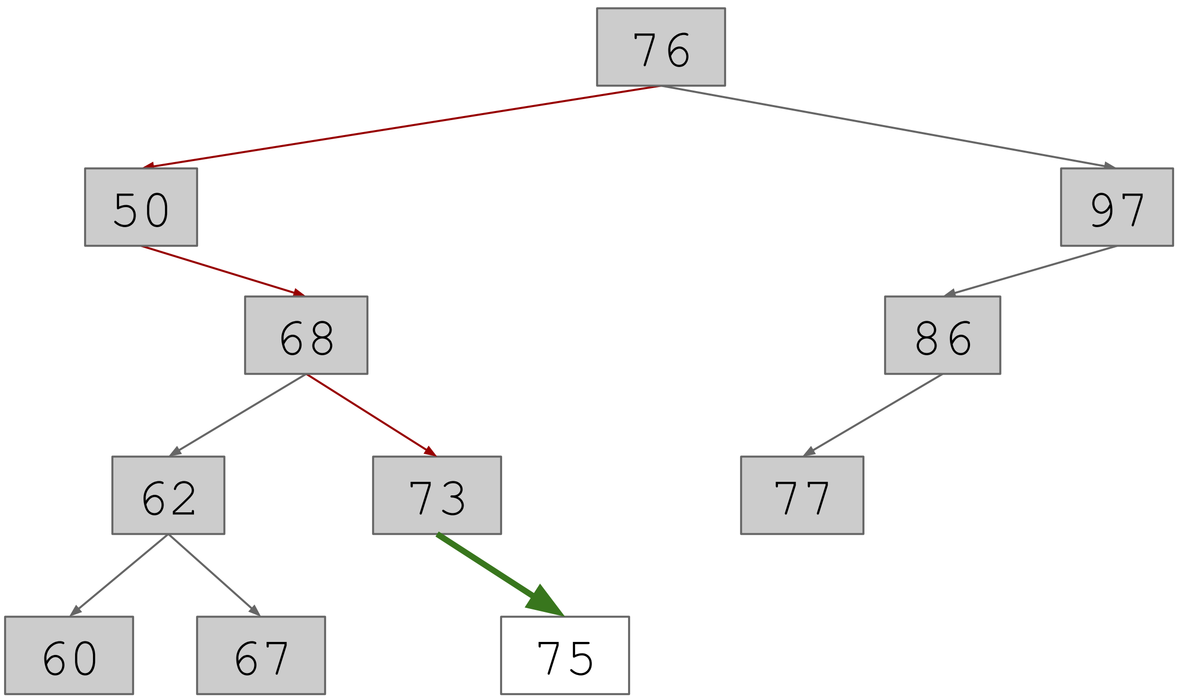
Searching for 75:



Binary Search Tree

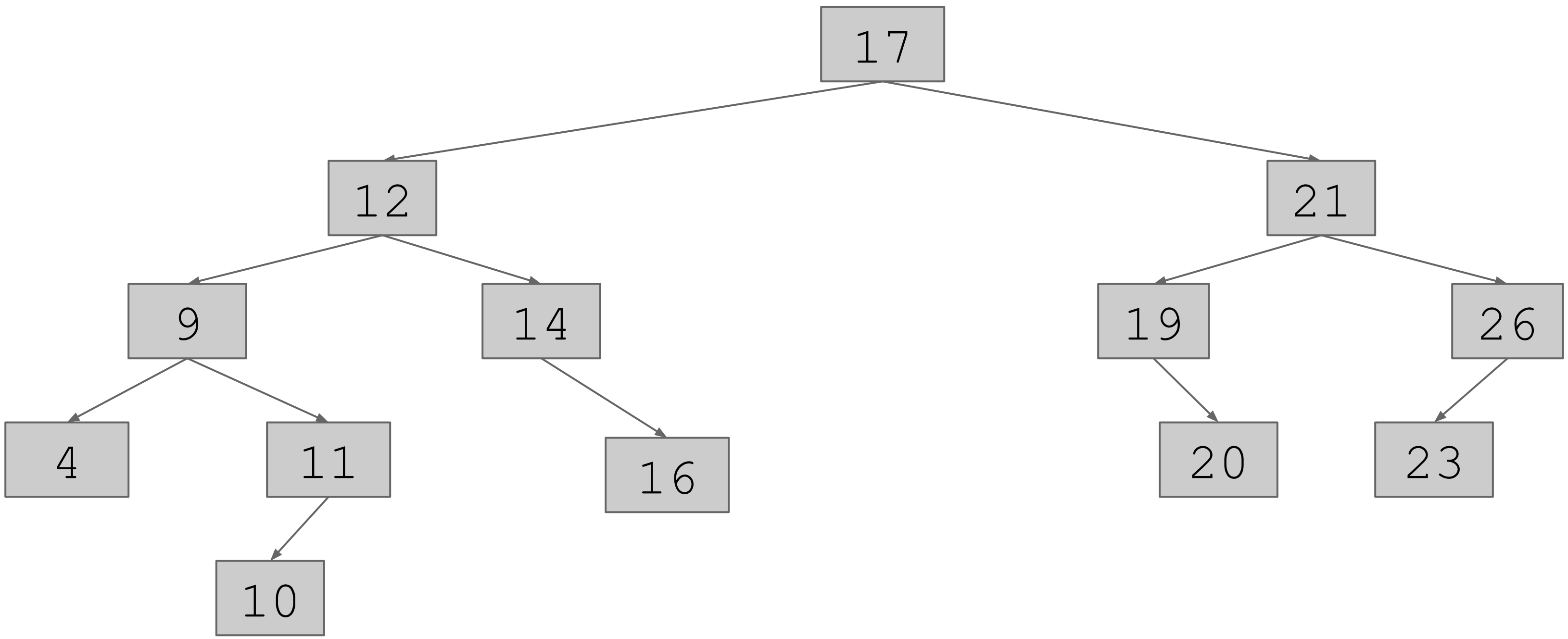
Insert in a binary search tree: travel to the position of the key, and if there is no node at that position then insert a new node.

Insert key 75:



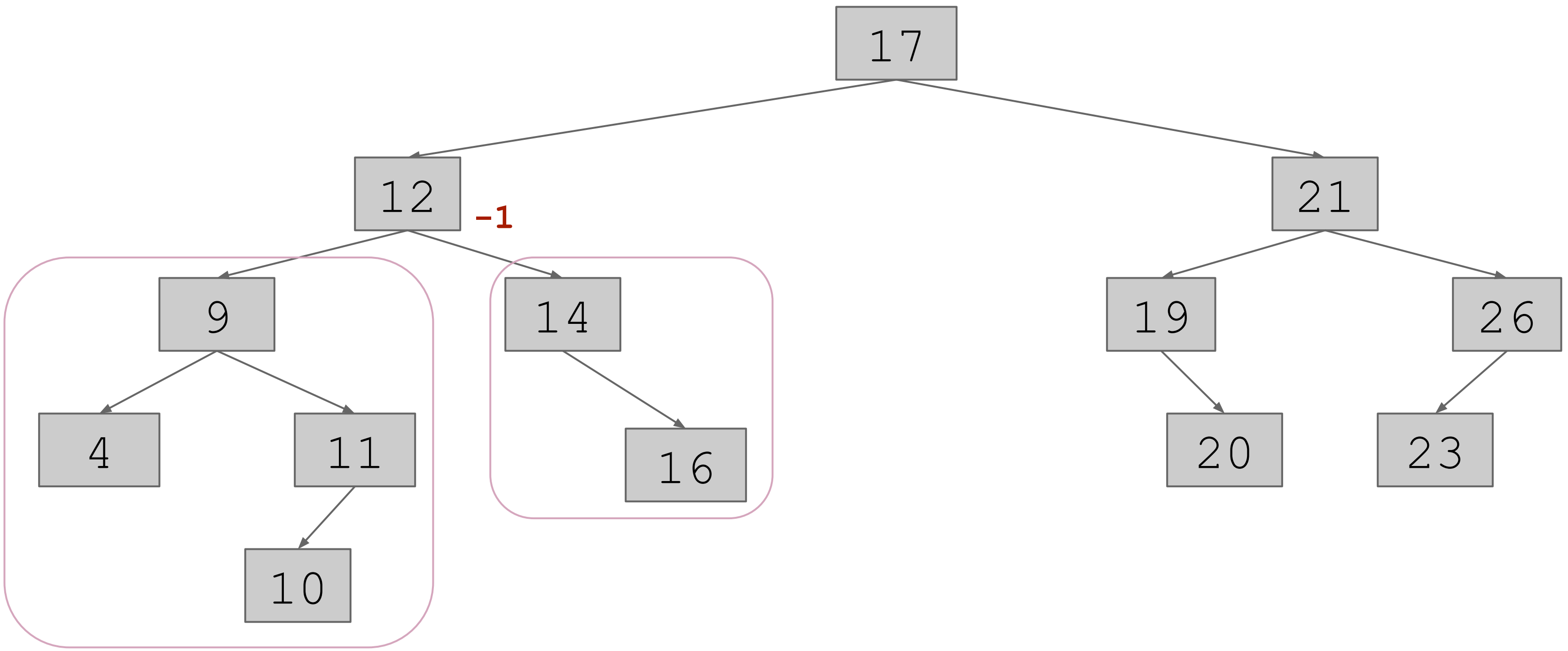
AVL Tree

AVL tree (invented by Adelson-Velsky and Landis): a **self-balancing** binary search tree.



AVL Tree

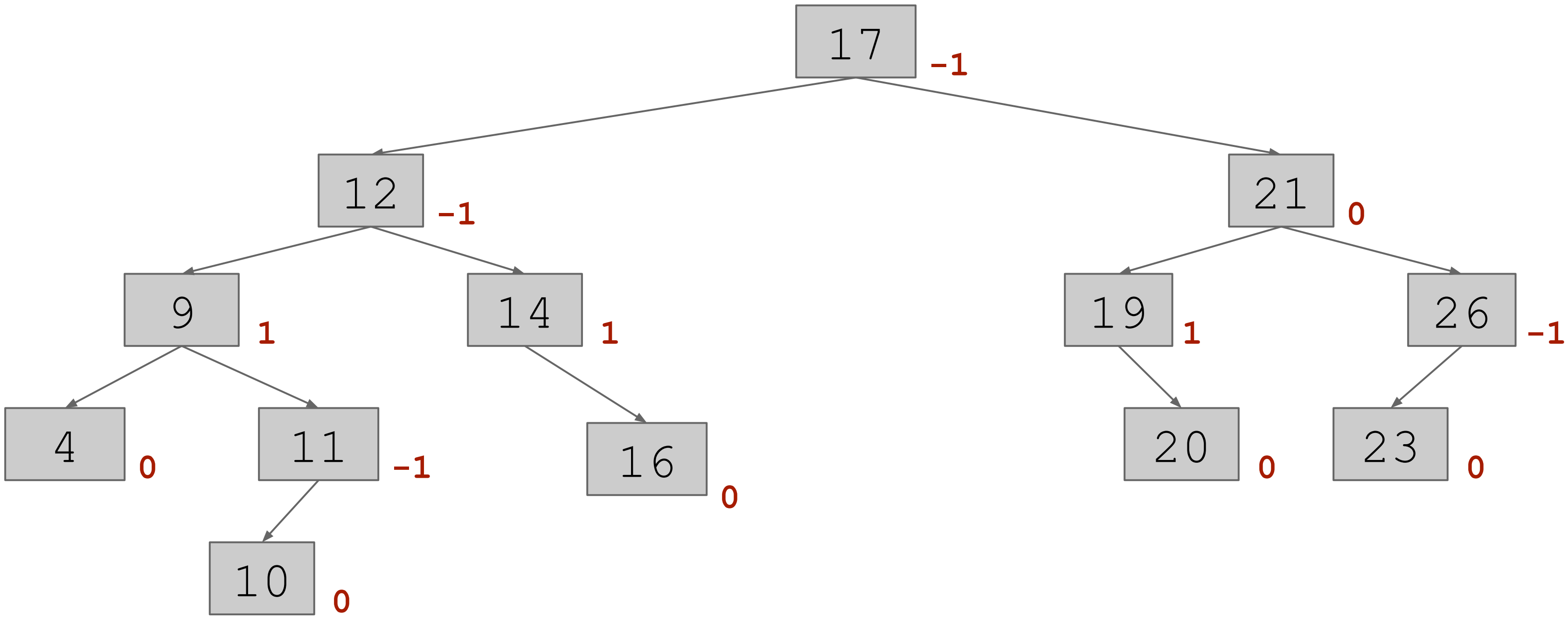
In an AVL tree, the heights of the two child subtrees of any node differ by at most one.




```
Balance factor = height of right subtree
                 - height of left subtree
```

AVL Tree

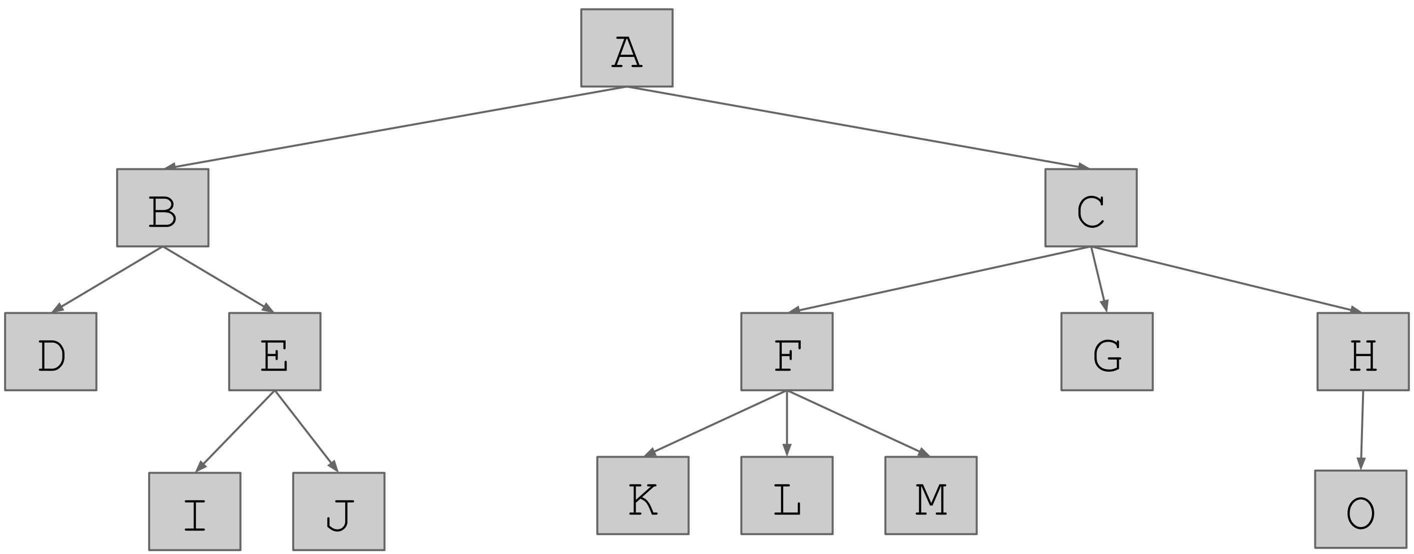
In an AVL tree, balance factor of any node = `-1, 0, 1`




```
Balance factor = height of right subtree
               - height of left subtree
```

N-ary Tree

N-ary tree: each node has at most N children



This is a ternary tree

References

- Python 3 documentation
  https://docs.python.org/3/
- NumPy Reference
  https://numpy.org/doc/stable/reference/