Introduction to Tree Data Structure

Data Structures CS218

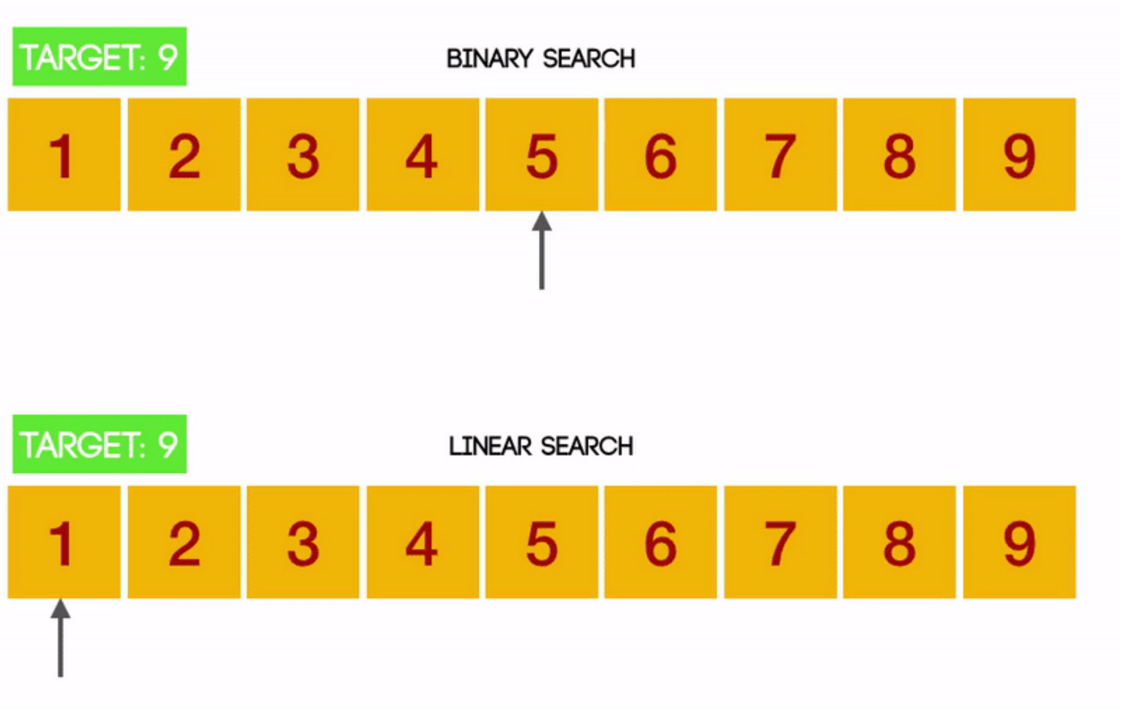
Outline

In this topic, we will cover:

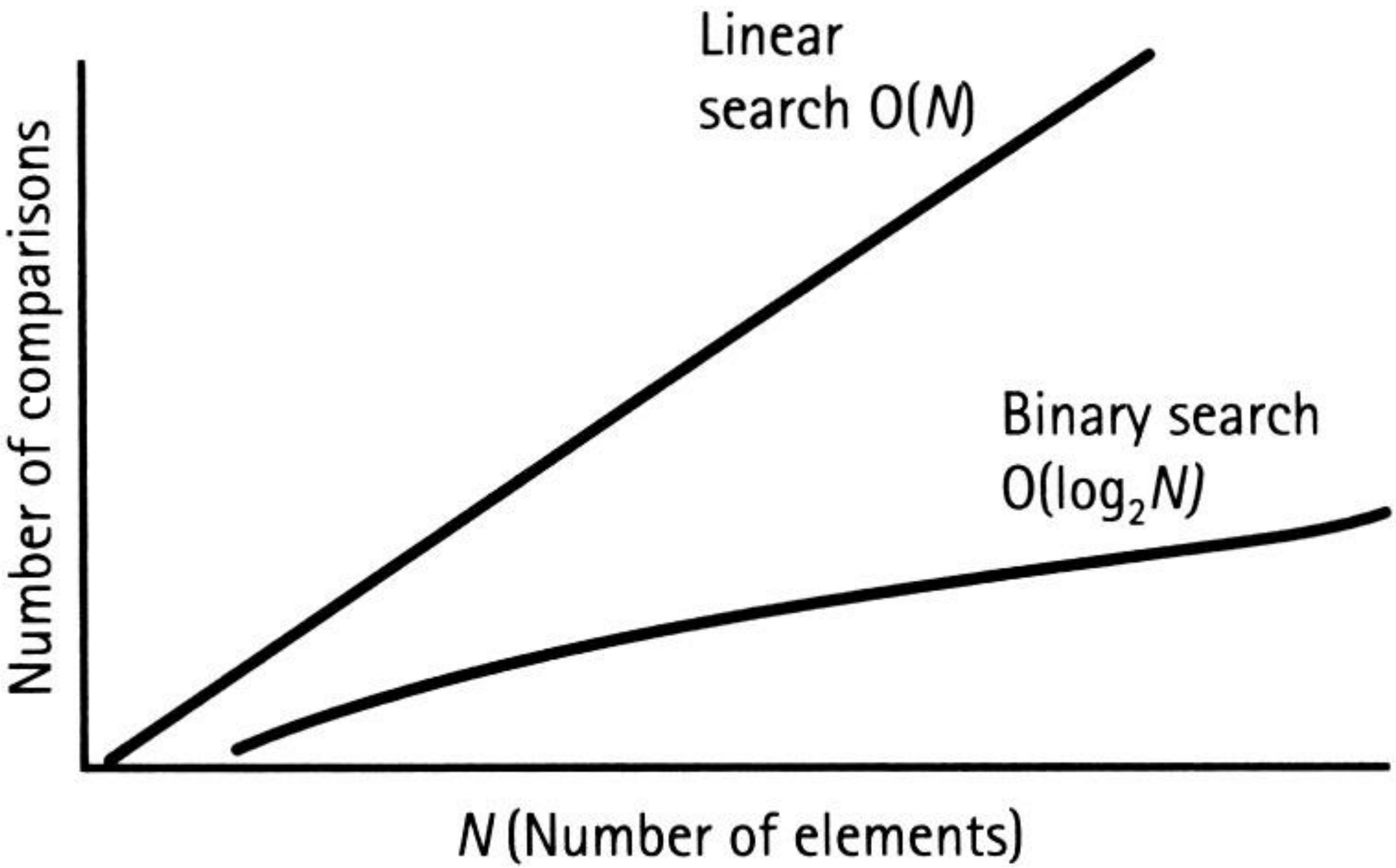
- From Linear to Non-Linear data structure
- Definition of a tree data structure and its components
- Concepts of:
  - Root, internal, and leaf nodes
  - Parents, children, and siblings
  - Paths, path length, height, and depth
  - Ancestors and descendants
  - Ordered and unordered trees
  - Subtrees
- Examples

From Linear to Non-Linear data structure

- Linear Search
- Binary Search



Linear Vs Binary Search



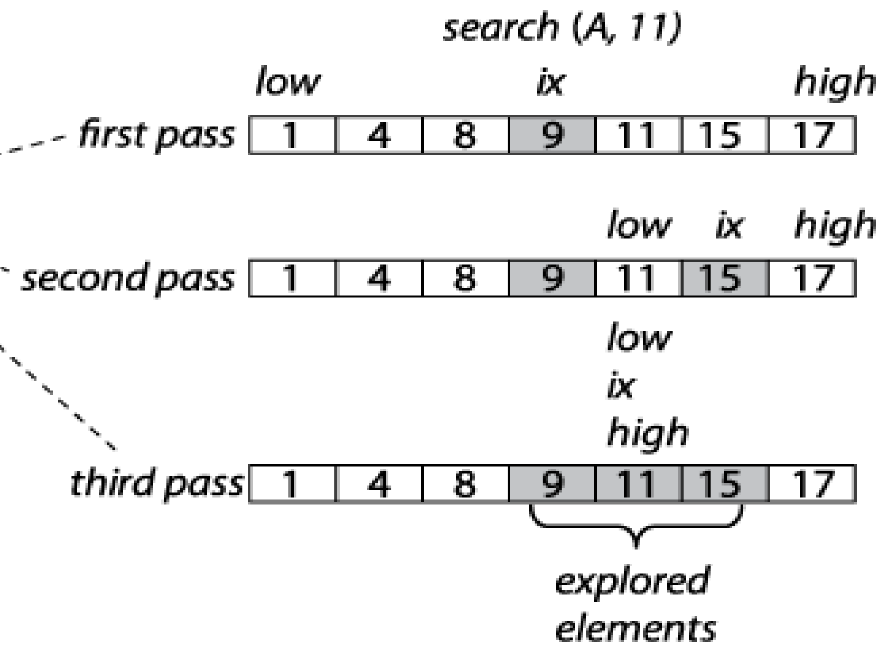
Binary Search Algorithm

Binary Search		
Best	Average	Worst
O (1)	O (log n)	O (log n)

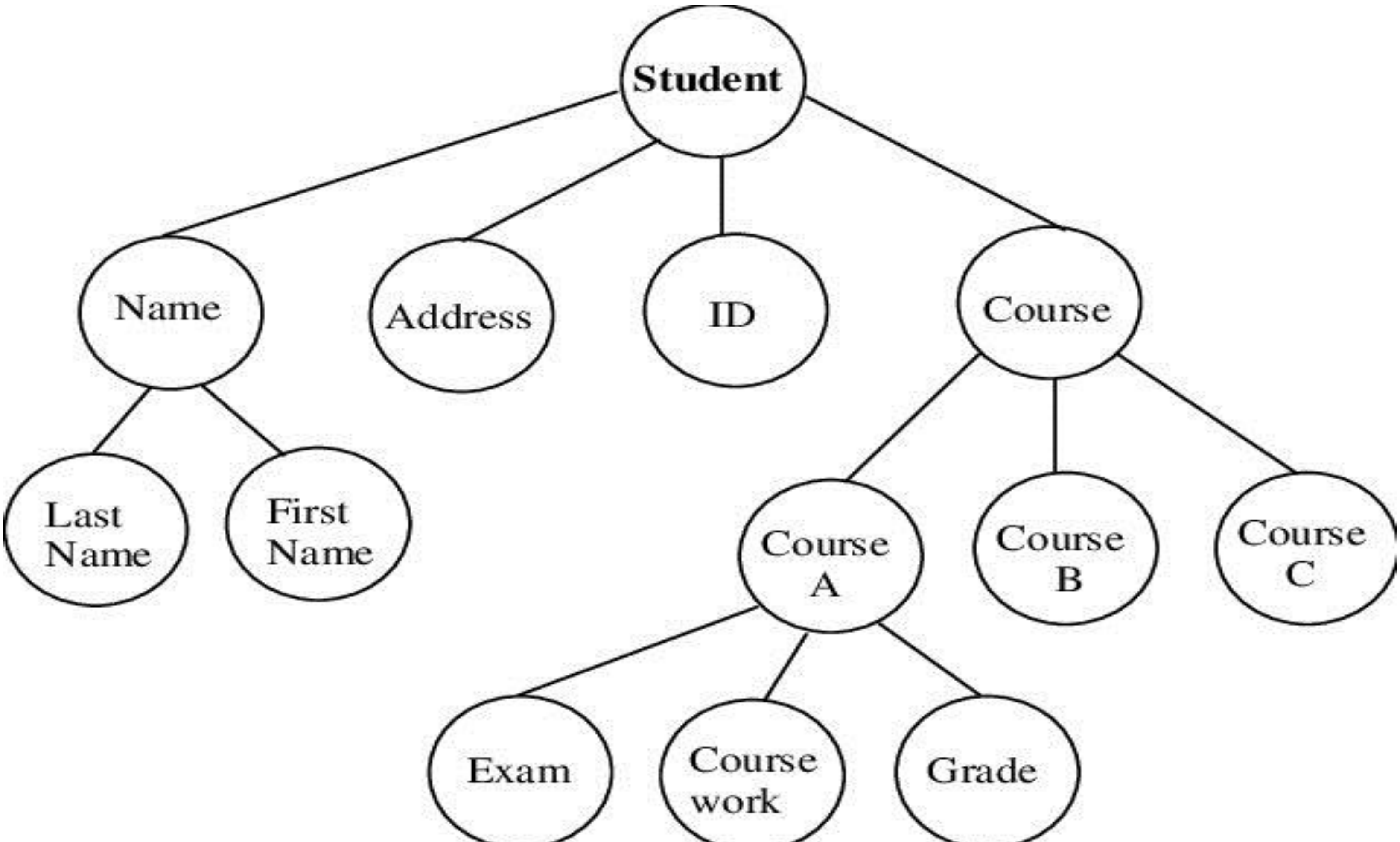
Array

Divide and Conquer

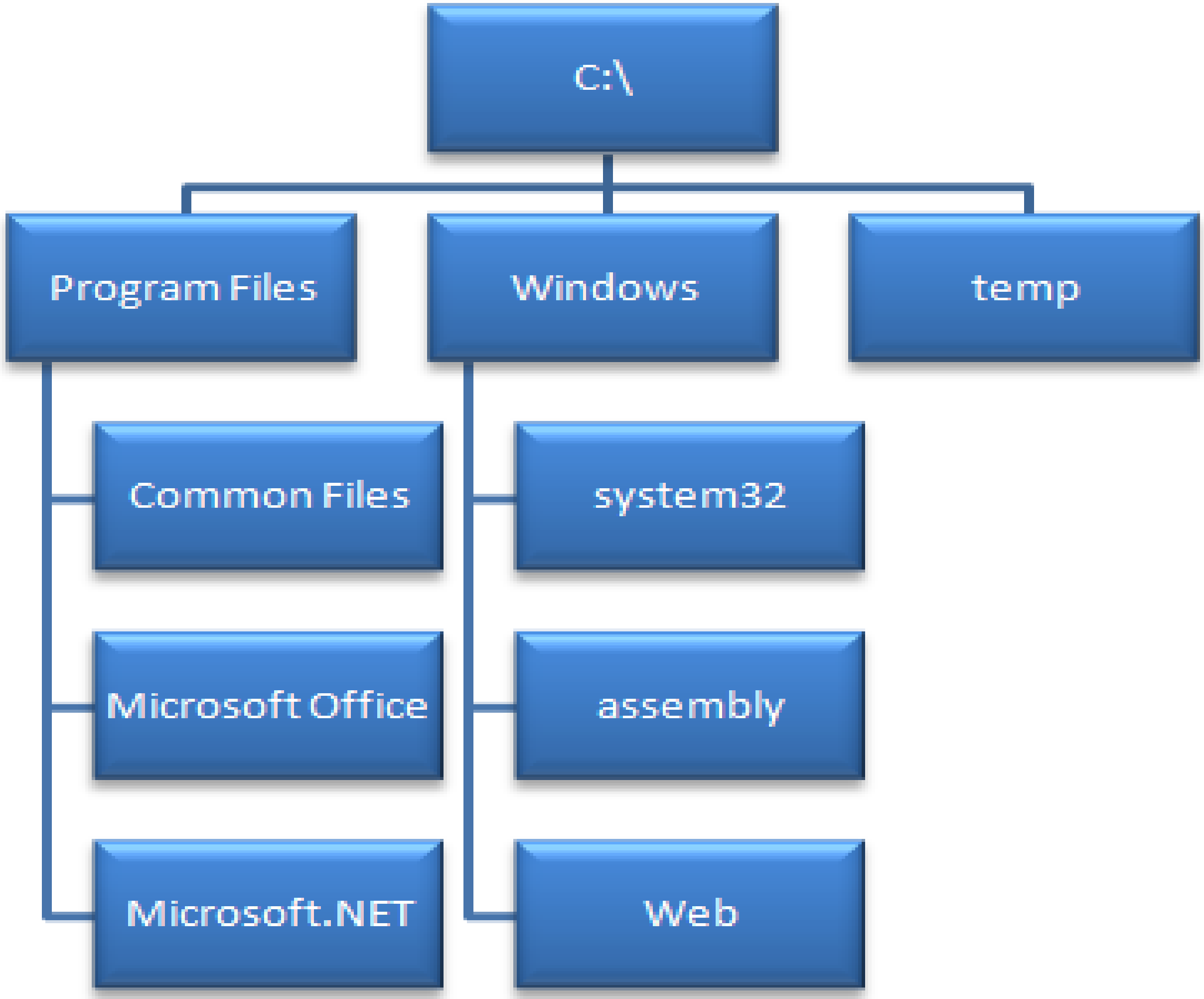
```
search (A, t)
    low = 0
    high = n −1
    while (low ≤ high) do
       ix = (low + high)/2
       if (t = A[ix]) then
           return true
       else if (t < A[ix]) then
           high = ix − 1
       else low = ix + 1
    return false
end
```



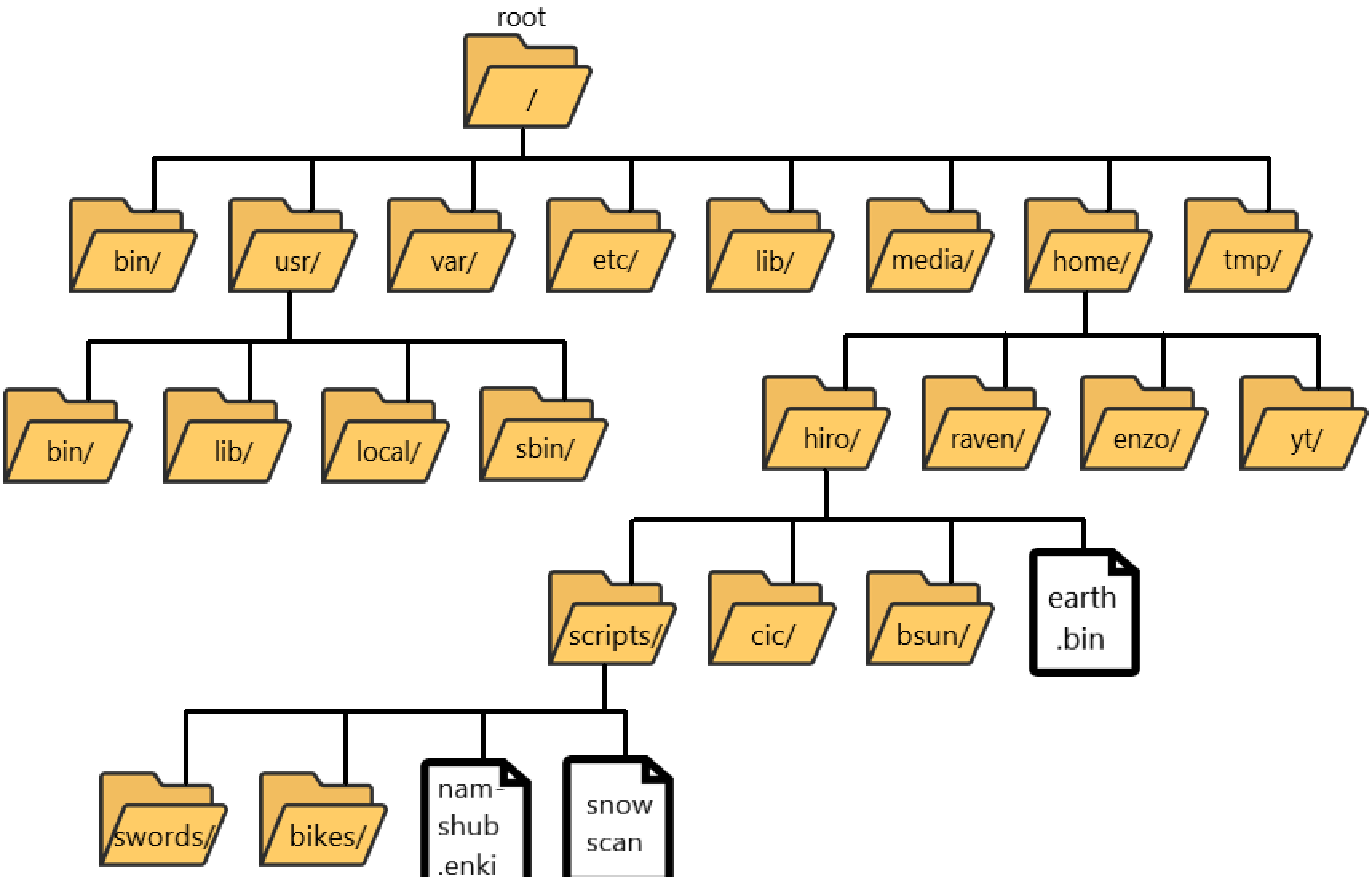
Why Trees Struture?



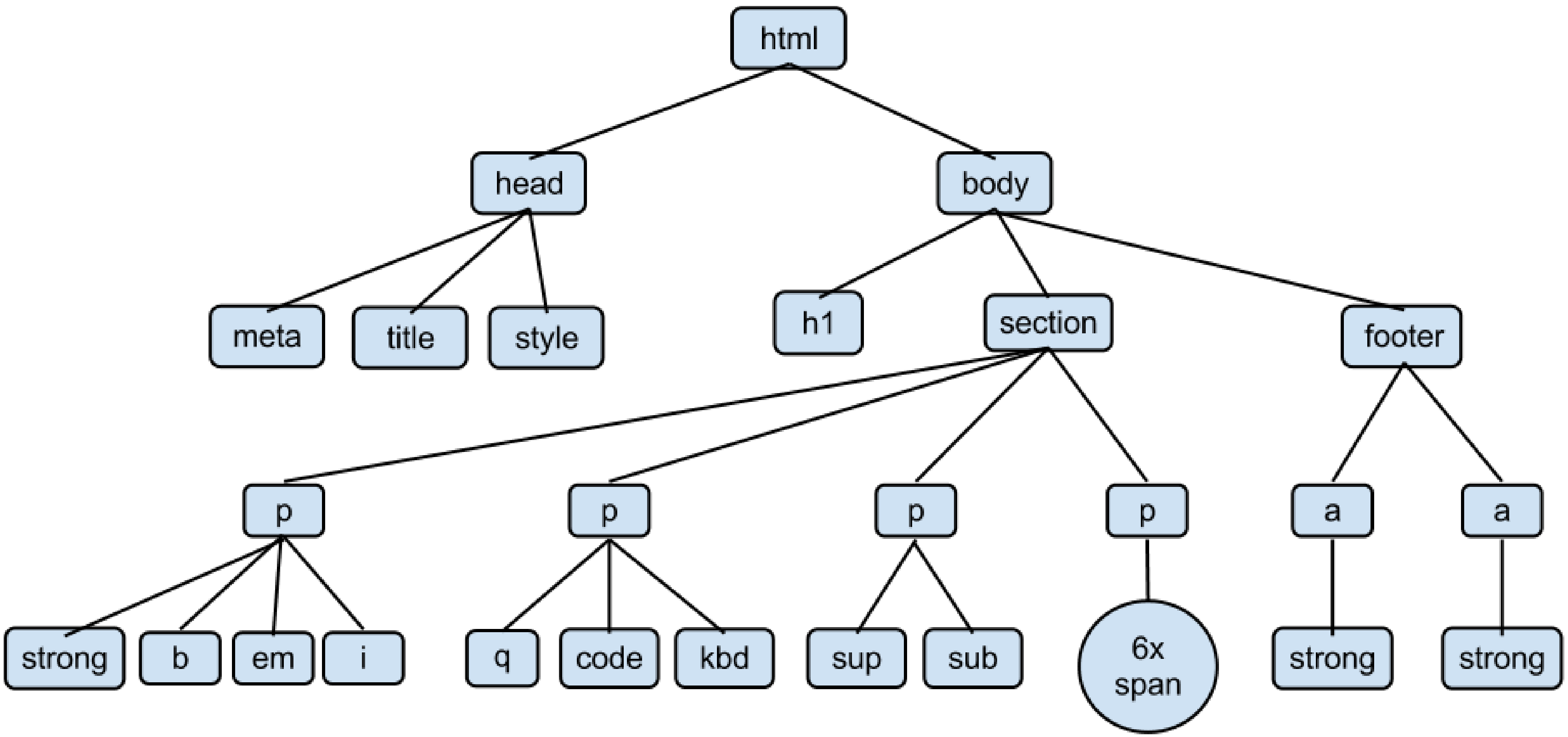
Why Trees Structure?



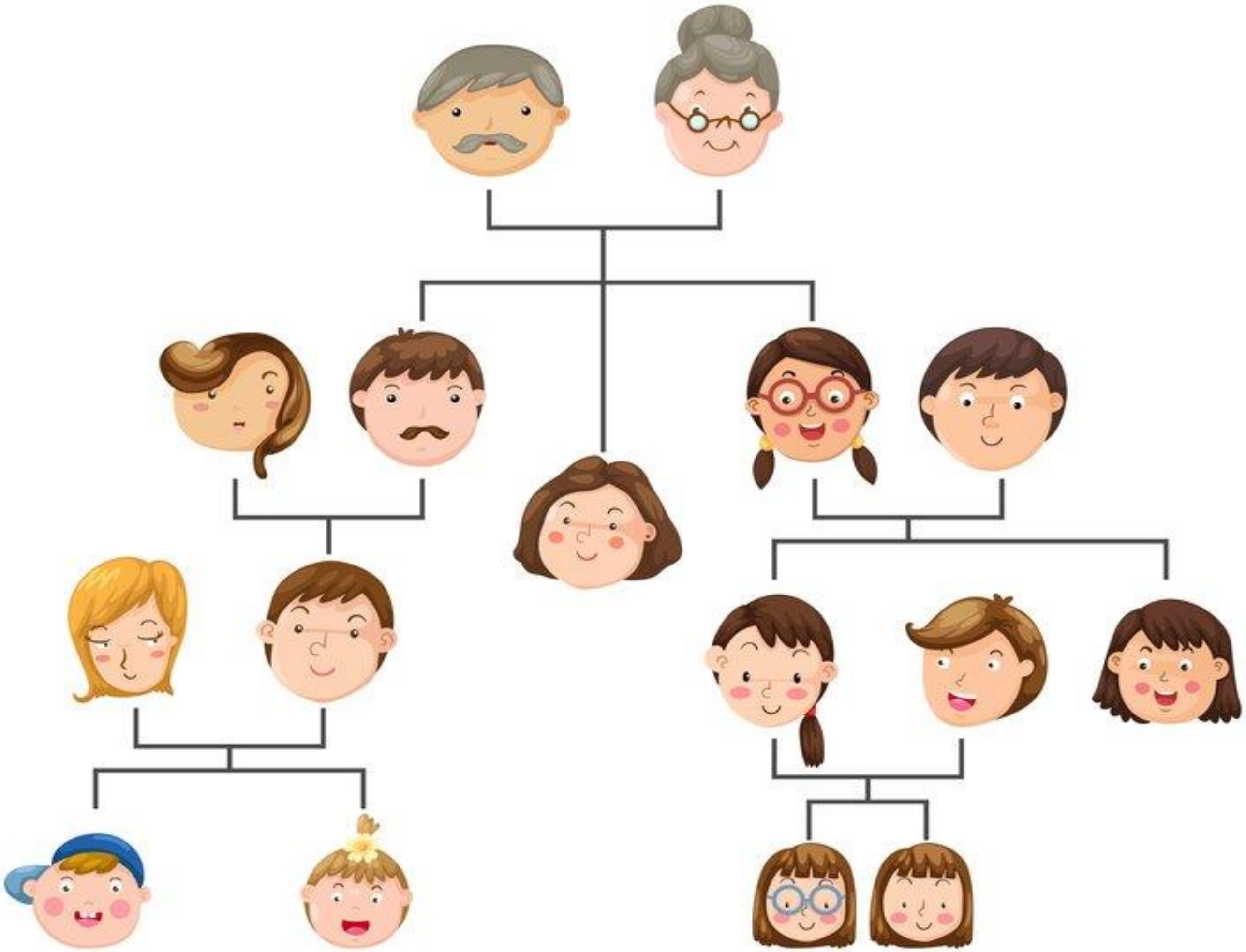
Why Trees Structure?



Why Trees Structure?



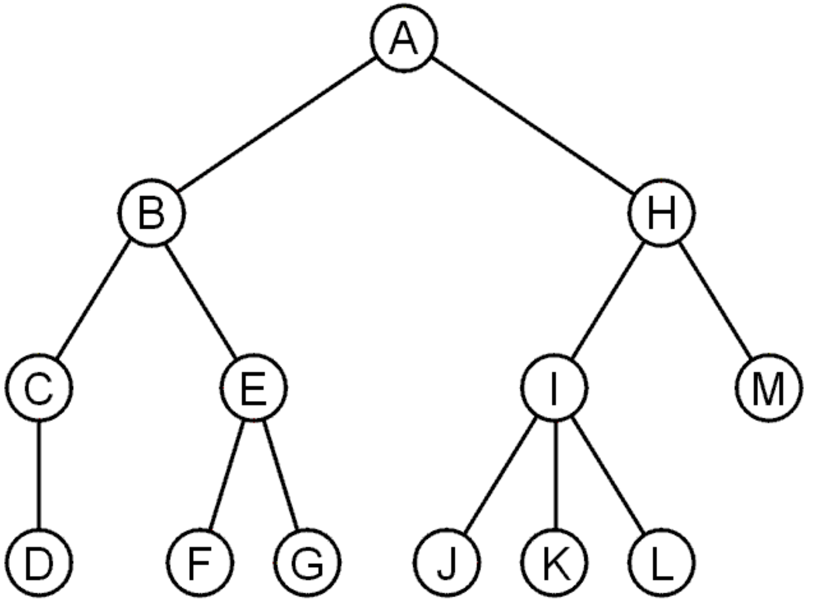
Why Trees Structure?



Trees

A rooted tree data structure stores information in *nodes*

- Similar to linked lists:
  - There is a first node, or *root*
  - Each node has variable number of references to successors
  - Each node, other than the root, has exactly one node pointing to it



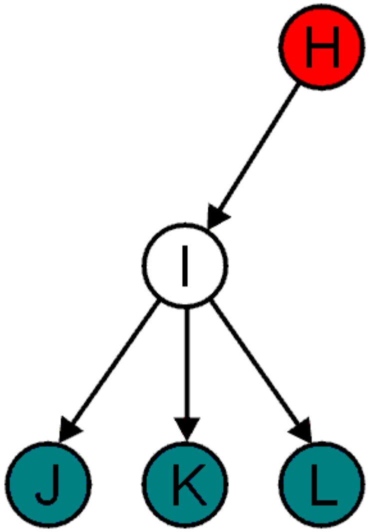
Terminology

All nodes will have zero or more child nodes or *children*

- I has three children: J, K and L

For all nodes other than the root node, there is one parent node

- H is the parent I



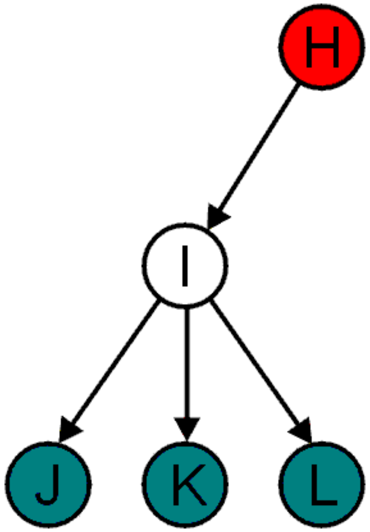
Terminology

The *degree* of a node is defined as the number of its children:

deg(I) = 3

Nodes with the same parent are *siblings*

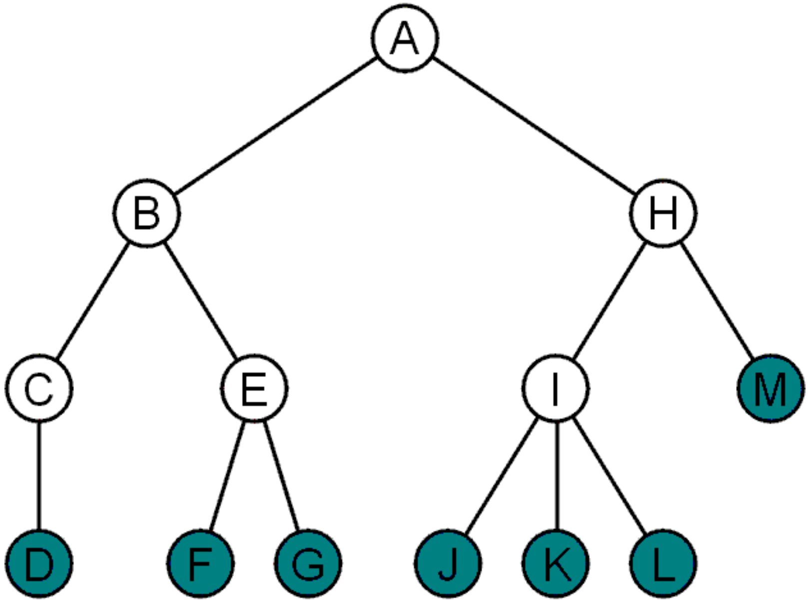
– J, K, and L are siblings



Terminology

Nodes with degree zero are also called *leaf nodes*

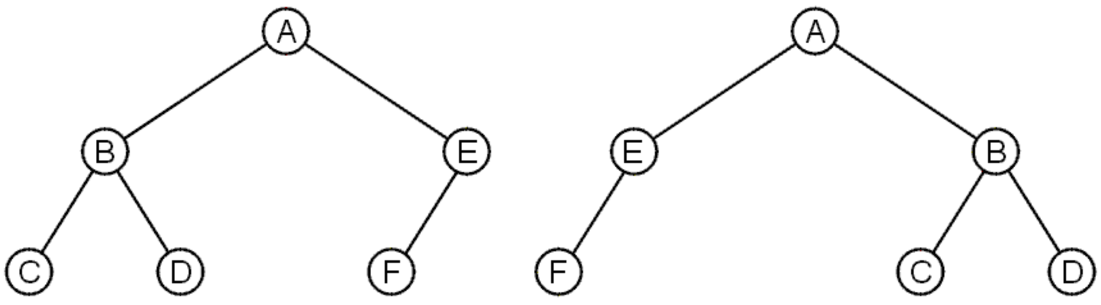
All other nodes are said to be *internal nodes*, that is, they are internal to the tree



Terminology

These trees are equal if the order of the children is ignored

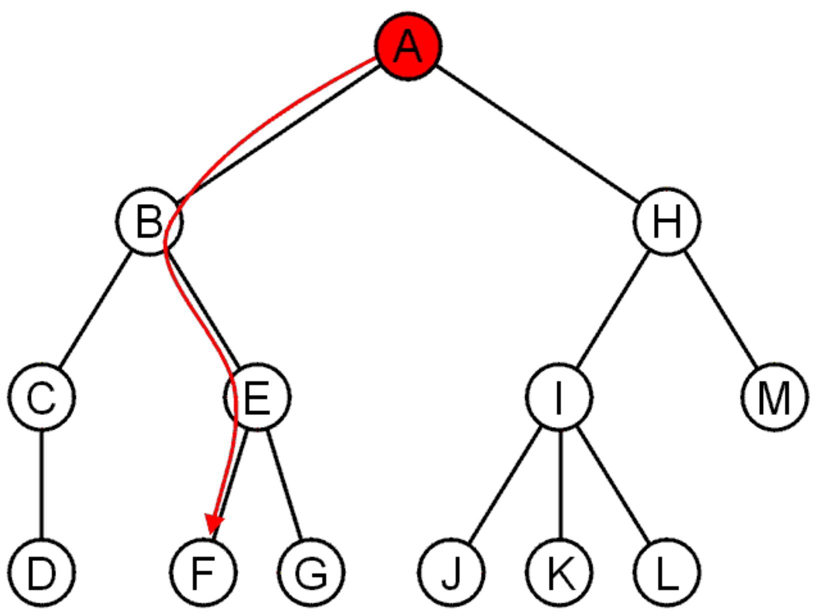
- *unordered trees*



They are different if order is relevant (*ordered trees*)

Terminology

The shape of a rooted tree gives a natural flow from the *root node*, or just *root*



Terminology

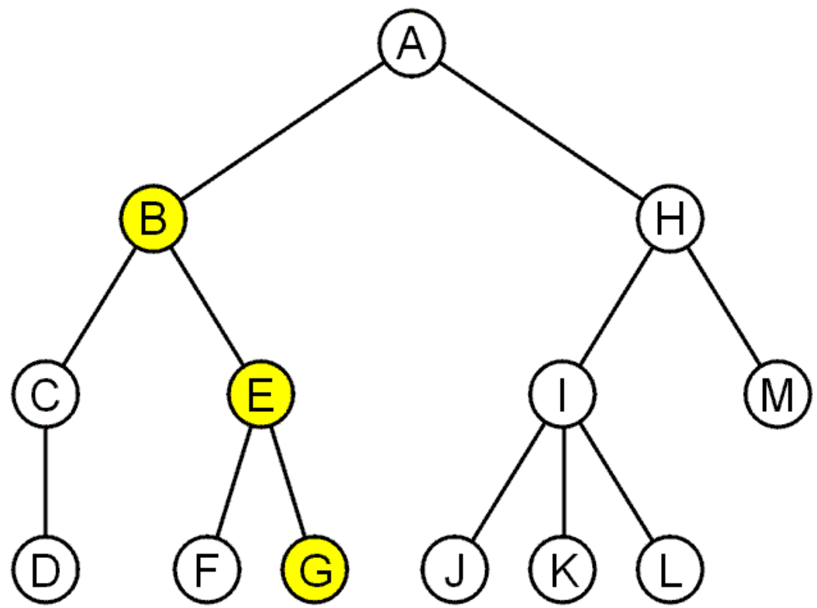
A path is a sequence of nodes

$(a_0, a_1, ..., a_n)$

where a_{k+1} is a child of a_k is

The length of this path is n

E.g., the path (B, E, G)
has length 2

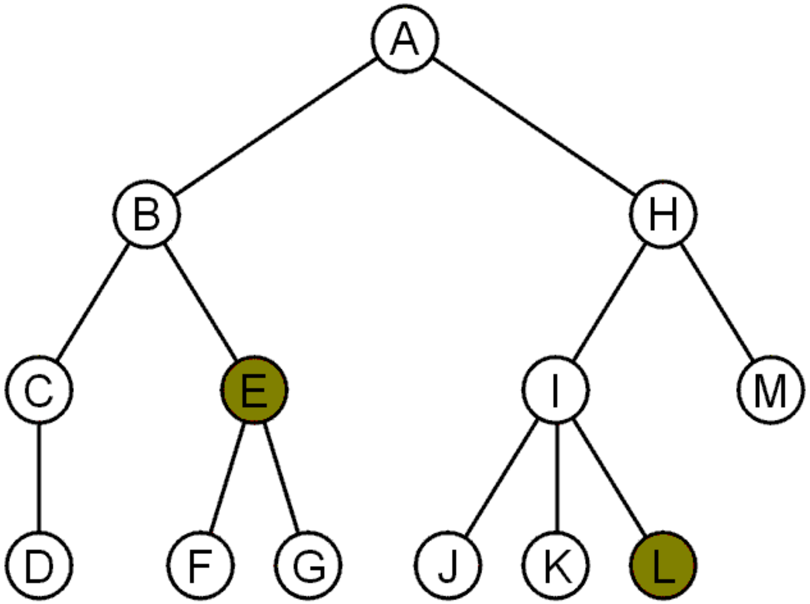


Terminology

For each node in a tree, there exists a unique path from the root node to that node

The length of this path is the *depth* of the node, *e*.*g*.,

- E has depth 2
- L has depth 3



Terminology

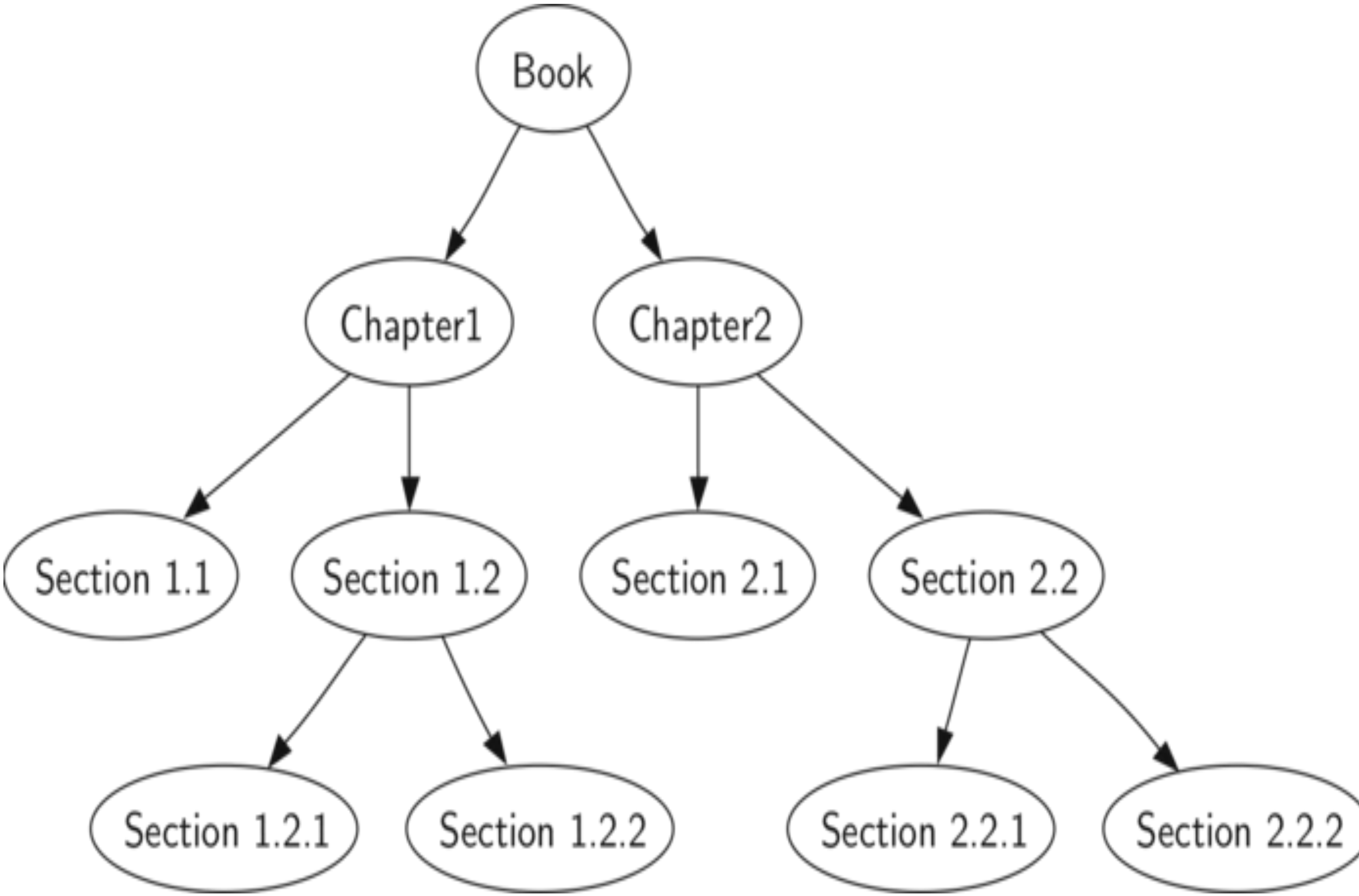
The *height* of a tree is defined as the maximum depth of any node within the tree

The height of a tree with one node is 0

– Just the root node

For convenience, we define the height of the empty tree to be -1

Terminology



Terminology

If a path exists from node a to node b:

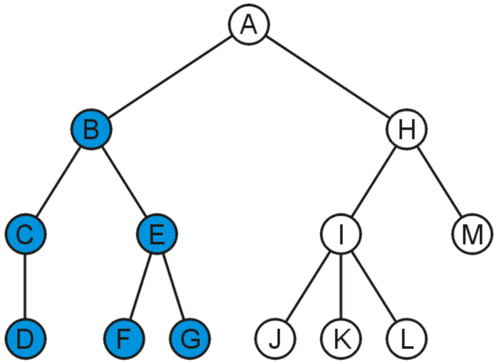
- a is an *ancestor* of b
- b is a *descendent* of a

Thus, a node is both an ancestor and a descendant of itself

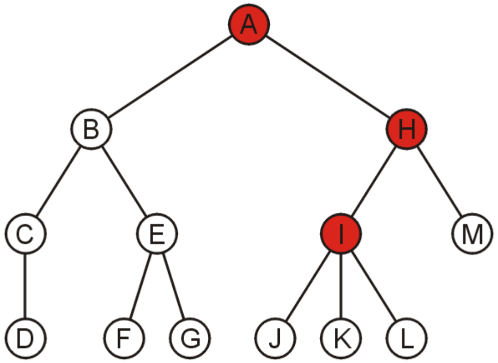
The root node is an ancestor of all nodes

Terminology

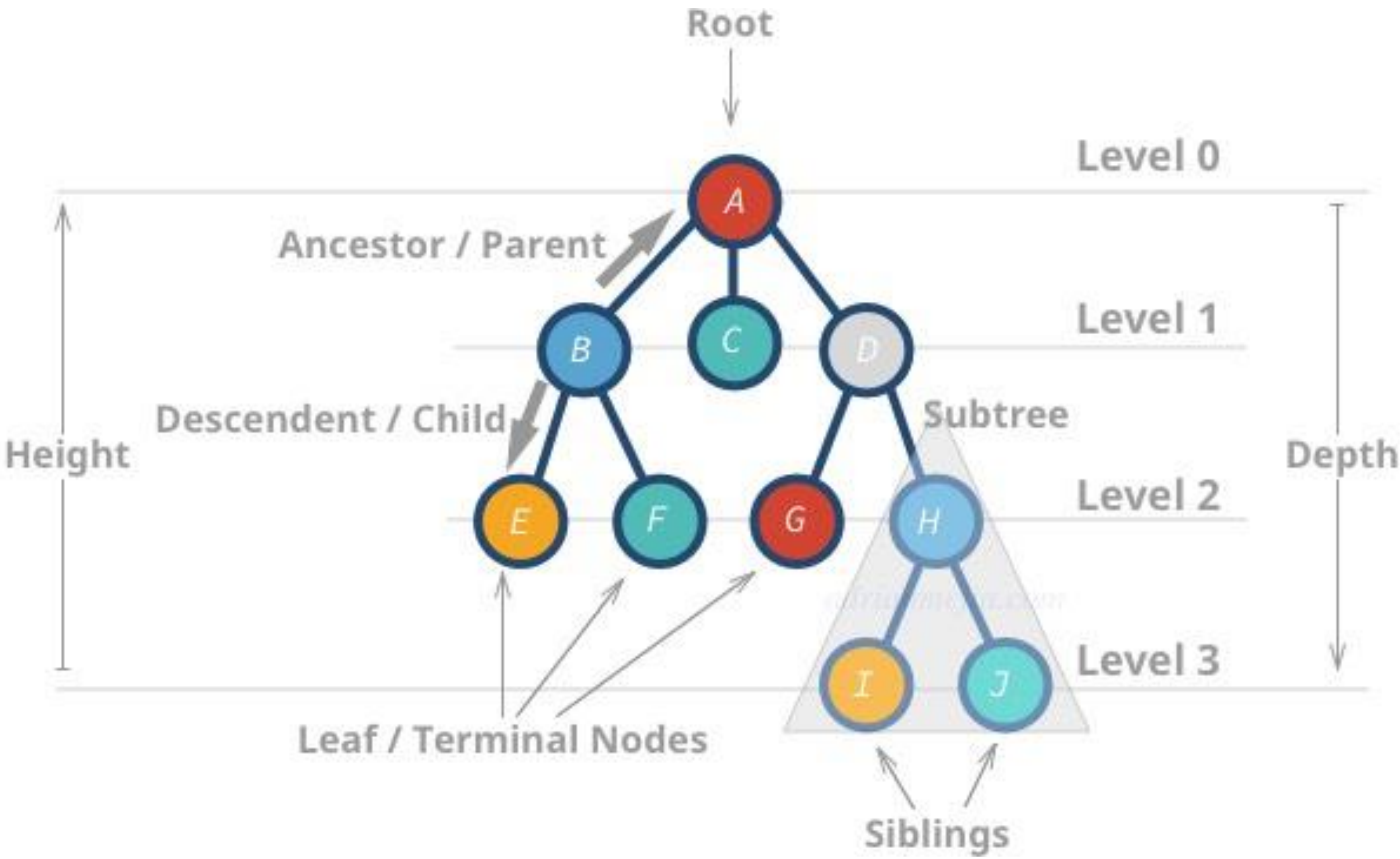
The descendants of node B are B, C, D, E, F, and G:



The ancestors of node I are I, H, and A:



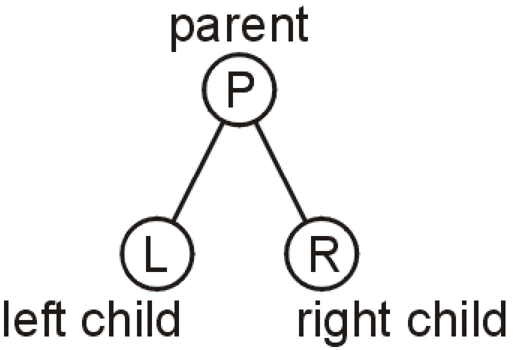
Summarized Terminologies



A Binary Tree

A binary tree is a restriction where each node has exactly two children:

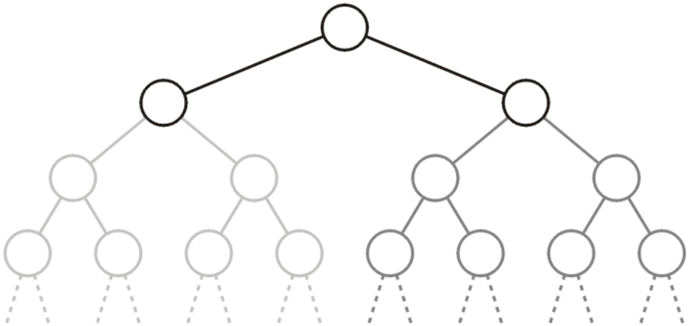
- Each child is either empty or another binary tree
- This restriction allows us to label the children as *left* and *right* subtrees



Binary Sub-trees

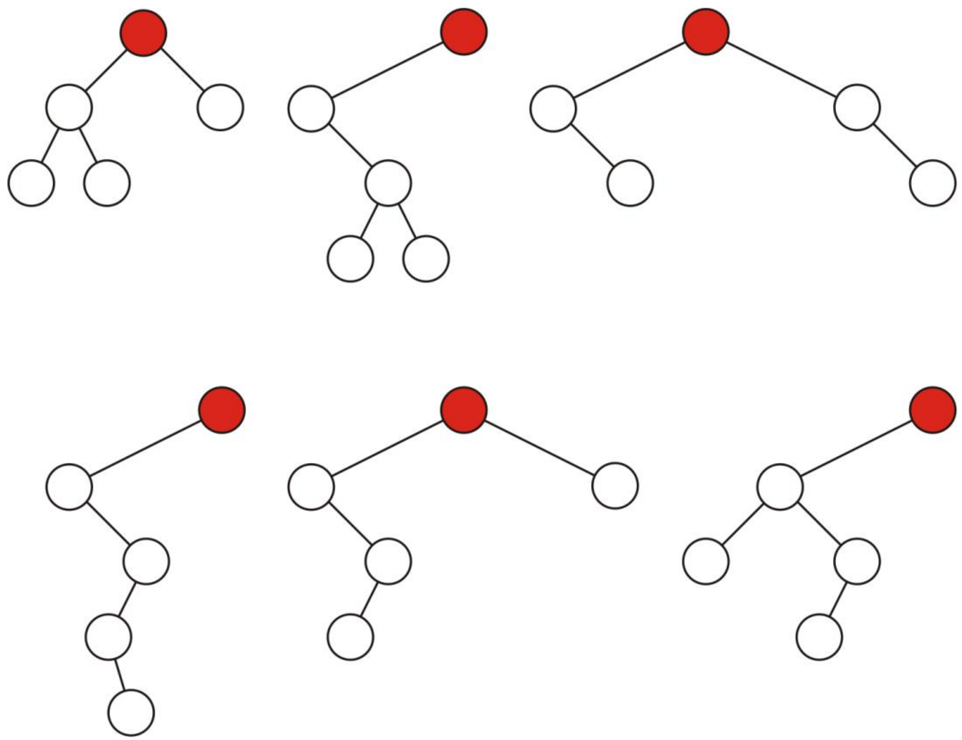
We will also refer to the two sub-trees as

- The left-hand sub-tree, and
- The right-hand sub-tree



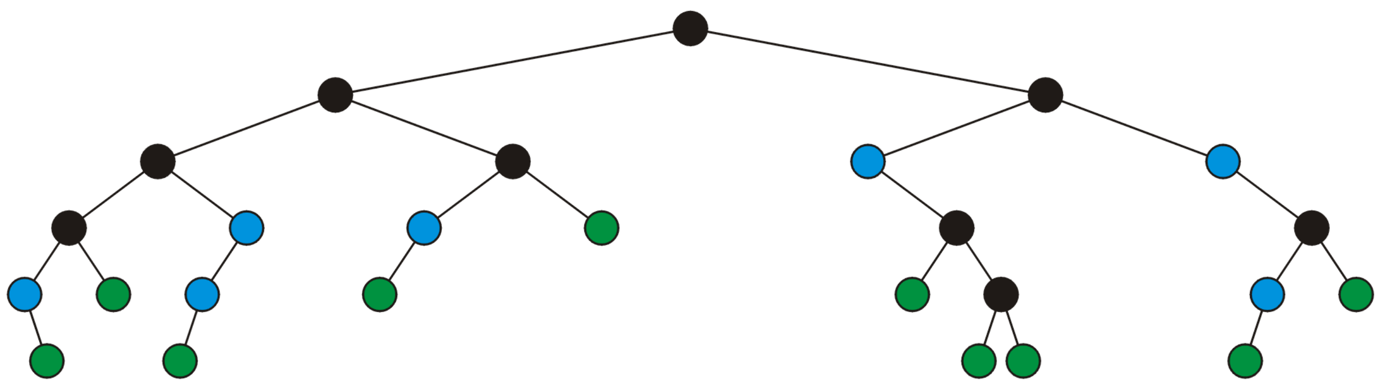
Sample Binary Trees

Sample variations on binary trees with five nodes:



Definition (Full Node)

A *full* node is a node where both the left and right sub-trees are non-empty trees

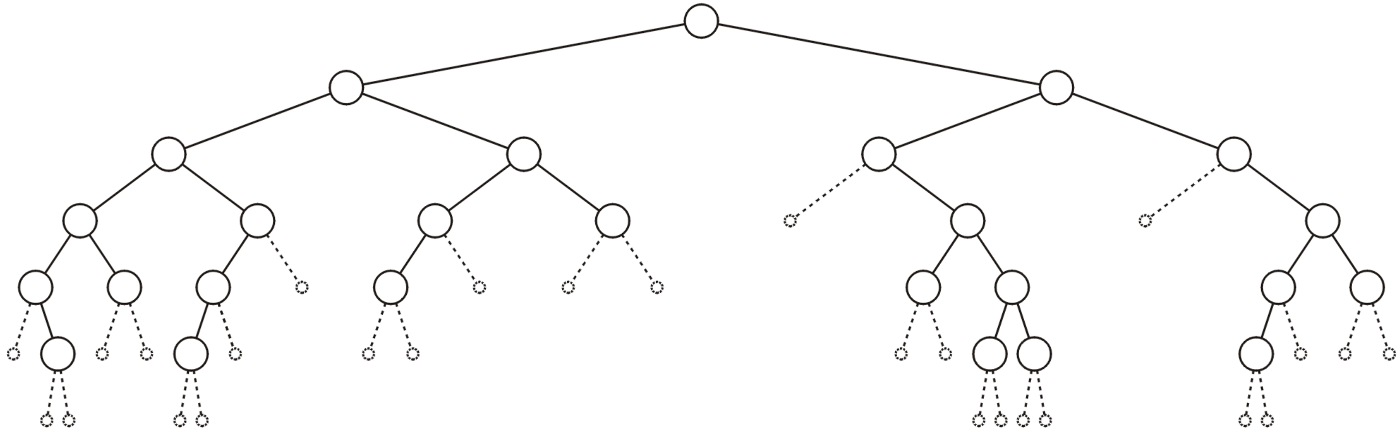


Legend:

full nodes neither leaf nodes

Definition(Empty Node)

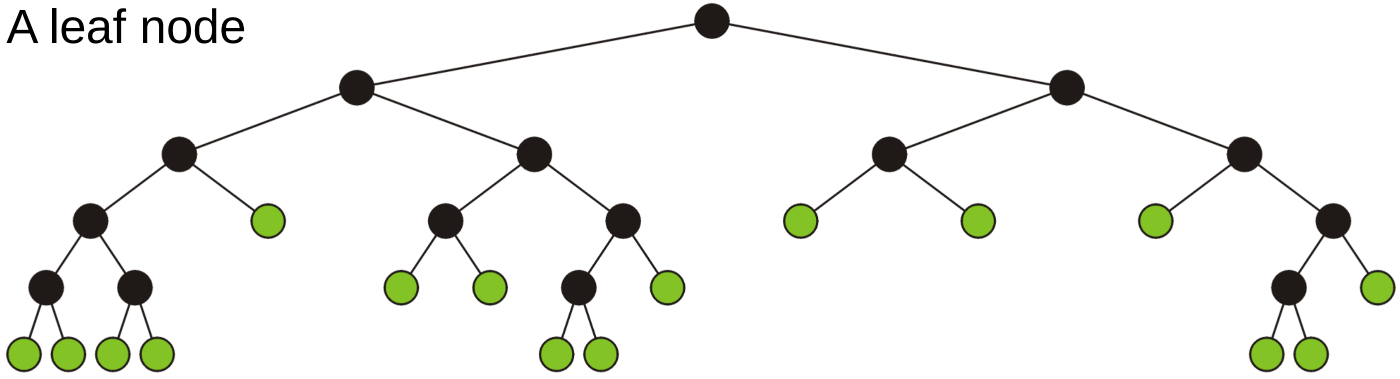
An *empty node* or a *null sub-tree* is any location where a new leaf node could be appended



Full Binary Tree

A *full binary tree* is where each node is:

- A full node, or
- A leaf node



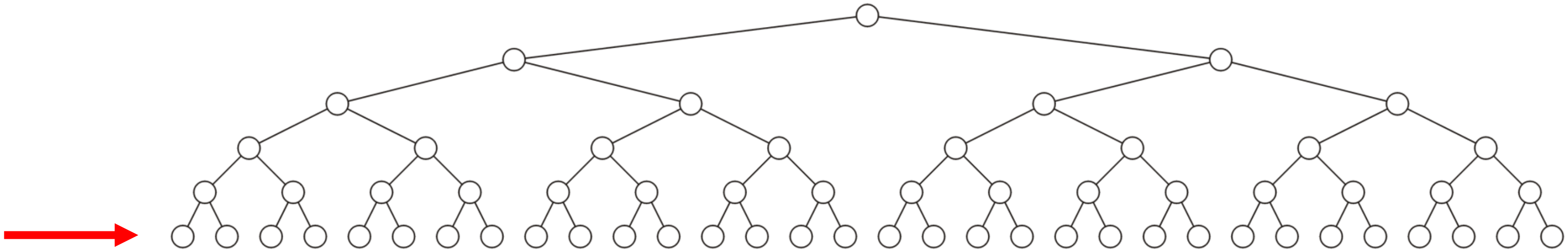
These have applications in

- Expression trees
- Huffman encoding

Perfect Binary Tree

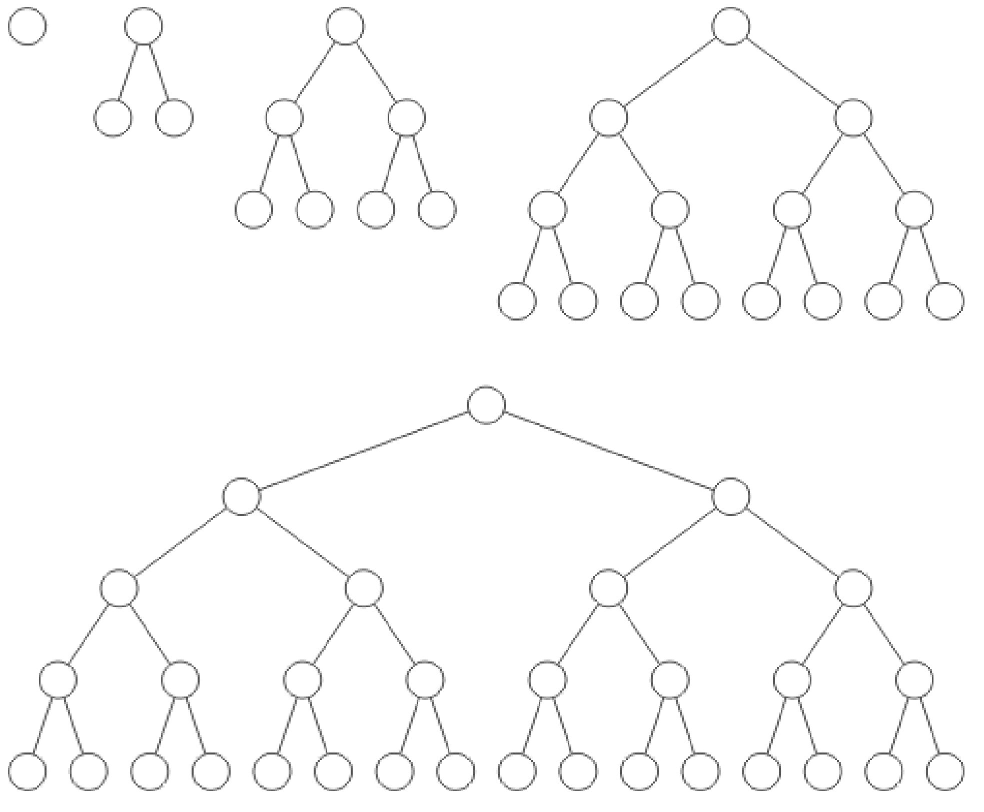
Standard definition:

– A perfect binary tree of height h is a binary tree where
  - All leaf nodes have the same depth h
  - All other nodes are full



Examples

Perfect binary trees of height $h = 0, 1, 2, 3$ and 4



Perfect Binary Trees

Perfect binary trees are considered to be the *ideal* case

- The height and average depth are both $\Theta(\ln(n))$

We will attempt to find trees which are as close as possible to perfect binary trees

One of the limitations of perfect binary trees is restricted number of nodes.

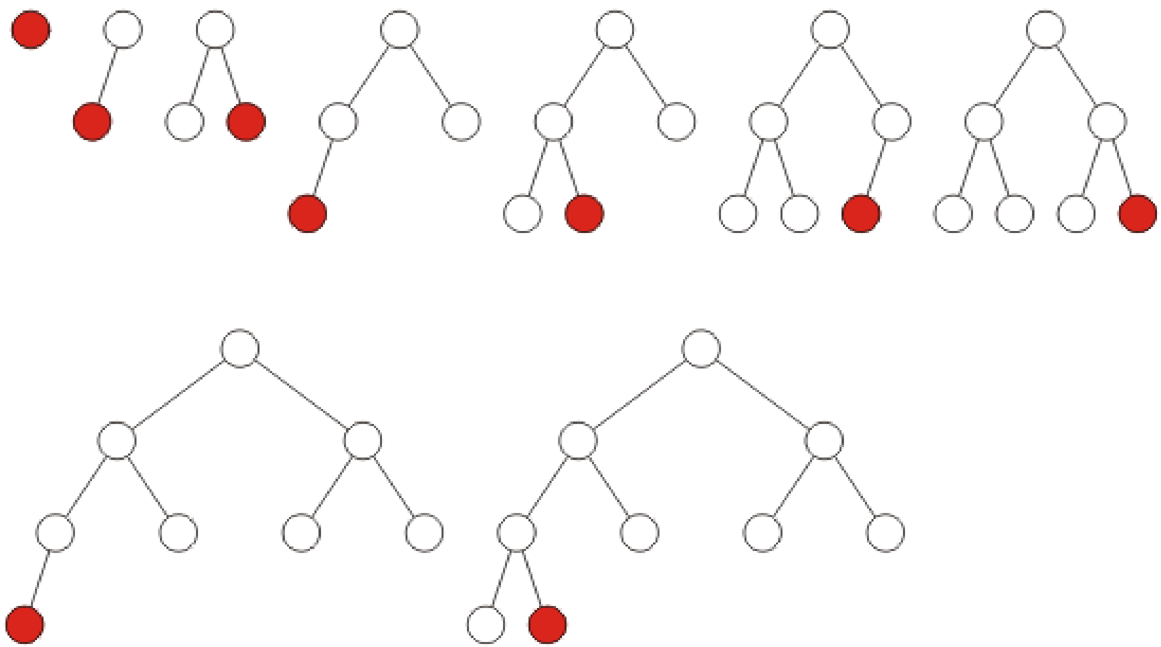
Complete Binary Trees

We require binary trees which are

- Similar to perfect binary trees, but
- Defined for all n

Complete Binary Trees

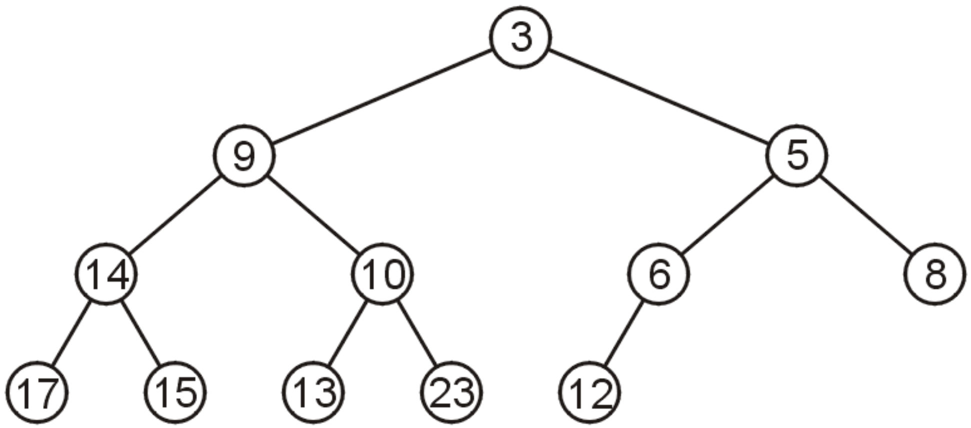
A complete binary tree filled at each depth from left to right:



Array Storage

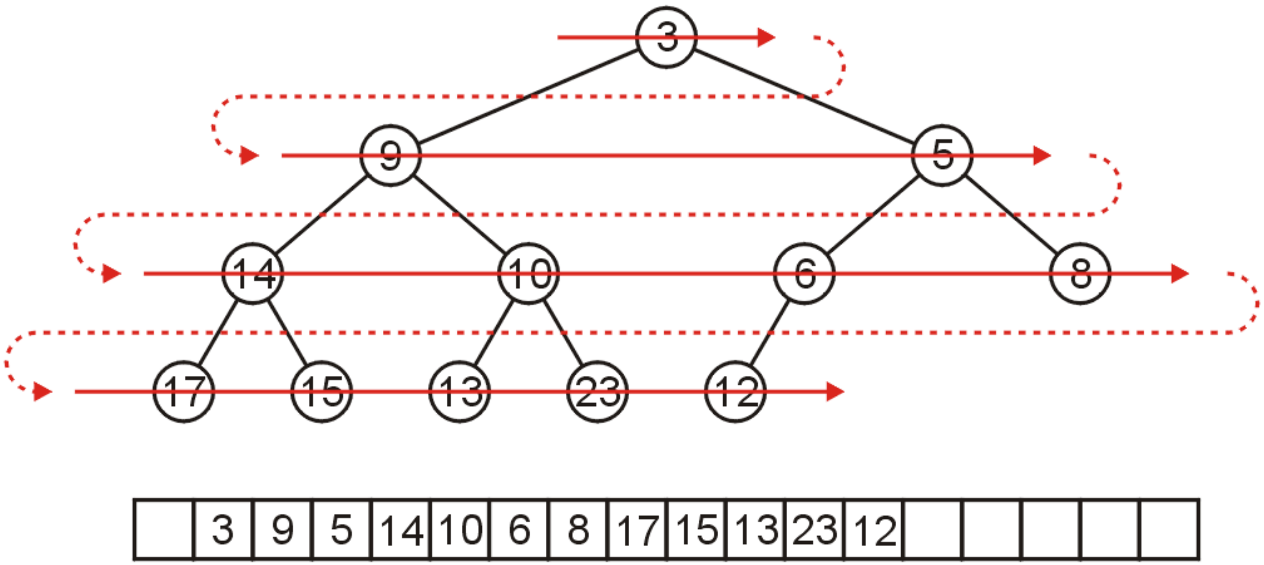
We are able to store a complete tree as an array

- Traverse the tree in breadth-first order, placing the entries into the array



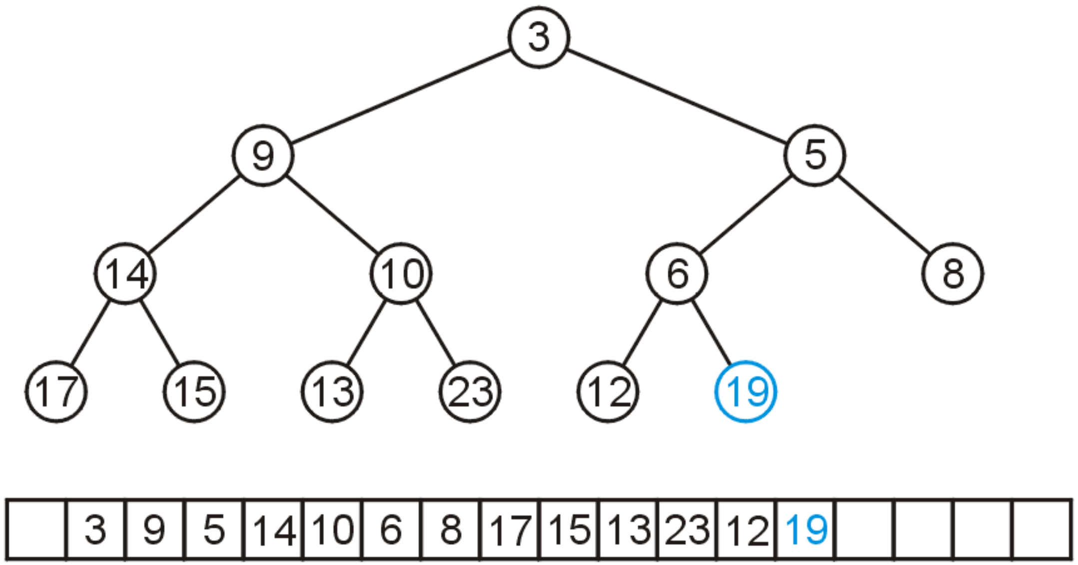
Array Storage

We can store this in an array after a quick traversal:



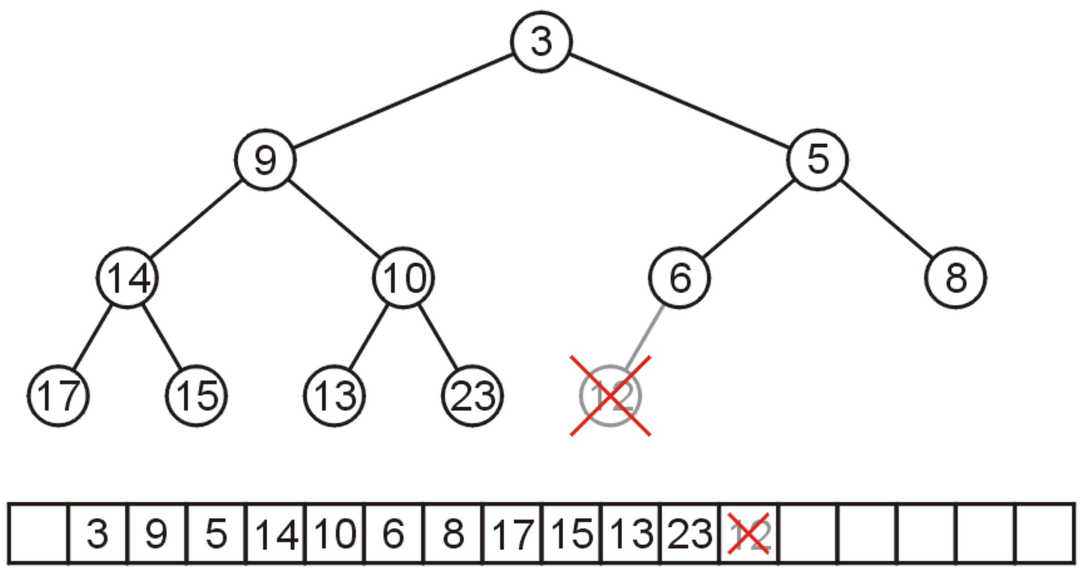
Array Storage

To insert another node while maintaining the complete-binary-tree structure, we must insert into the next array location



Array Storage

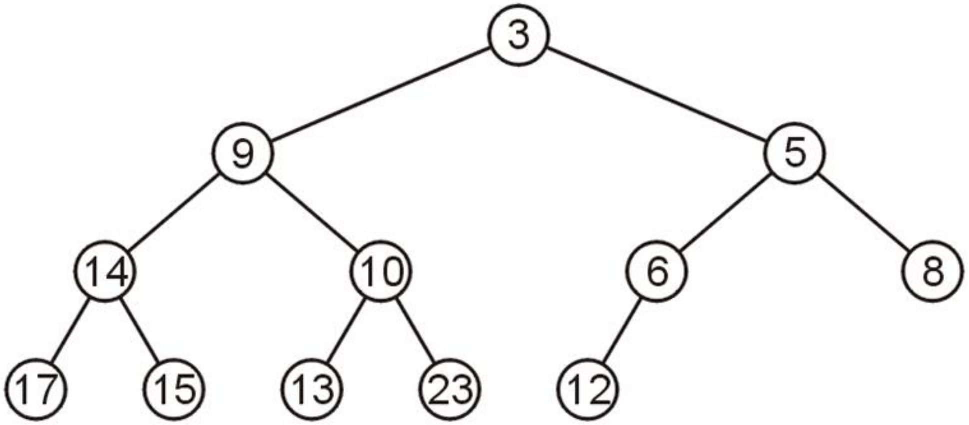
To remove a node while keeping the complete-tree structure, we must remove the last element in the array



Array Storage

Leaving the first entry blank yields a bonus:

- The children of the node with index k are in $2k$ and $2k + 1$
- The parent of node with index k is in $k \div 2$

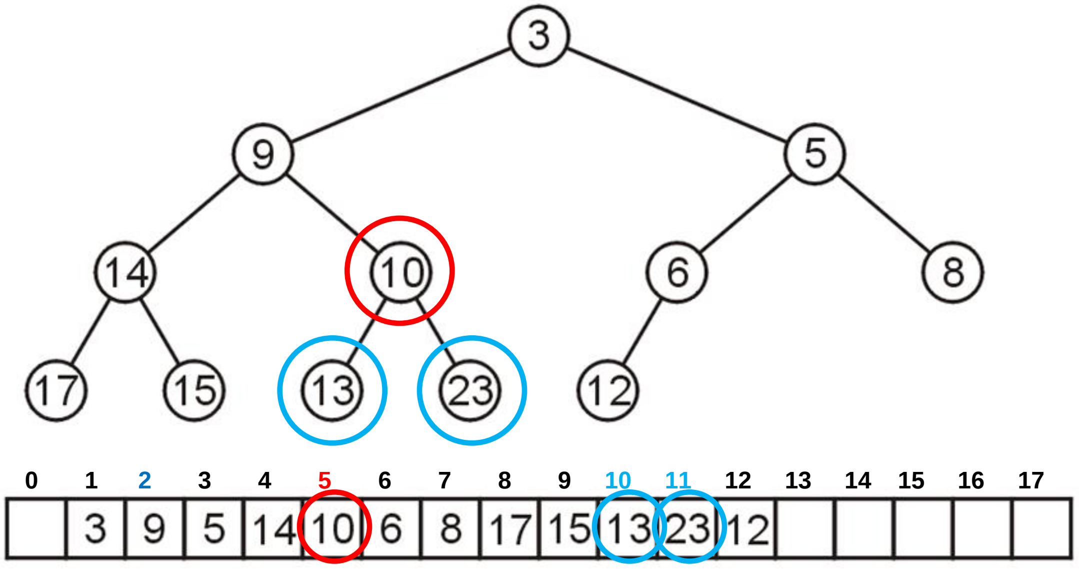


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	3	9	5	14	10	6	8	17	15	13	23	12					

Array Storage

For example, node 10 has index 5:

- Its children 13 and 23 have indices 10 and 11, respectively



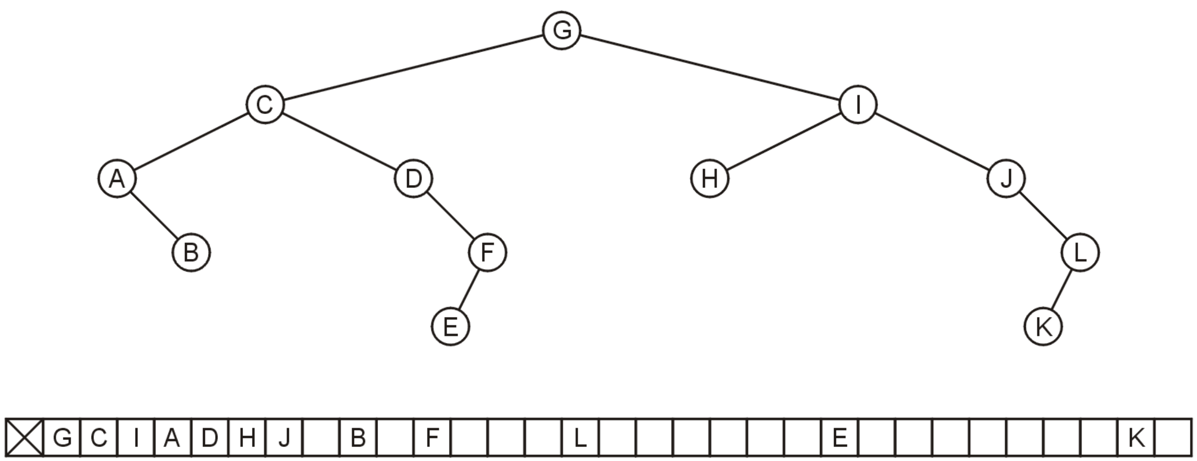
Array Storage

Question: why not store any tree as an array?

– There is a significant potential for a lot of wasted memory

Consider this tree with 12 nodes would require an array of size 32

– Adding a child to node K doubles the required memory



Summary

- In this topic, we have covered the concept of tree, binary tree, and types of binary tree
- We have also covered a compact array representation of a complete binary tree