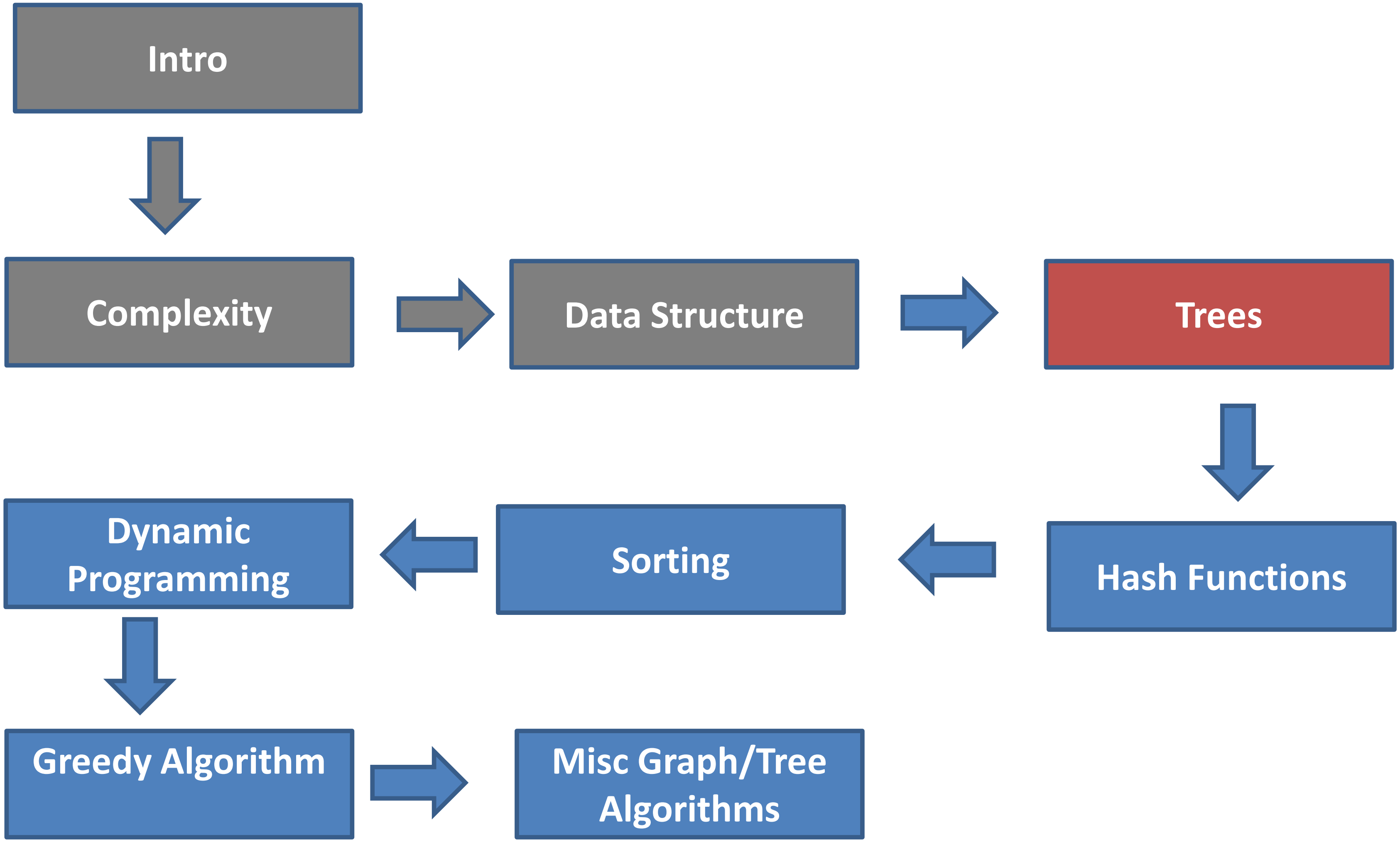
An Introduction to Algorithms

By

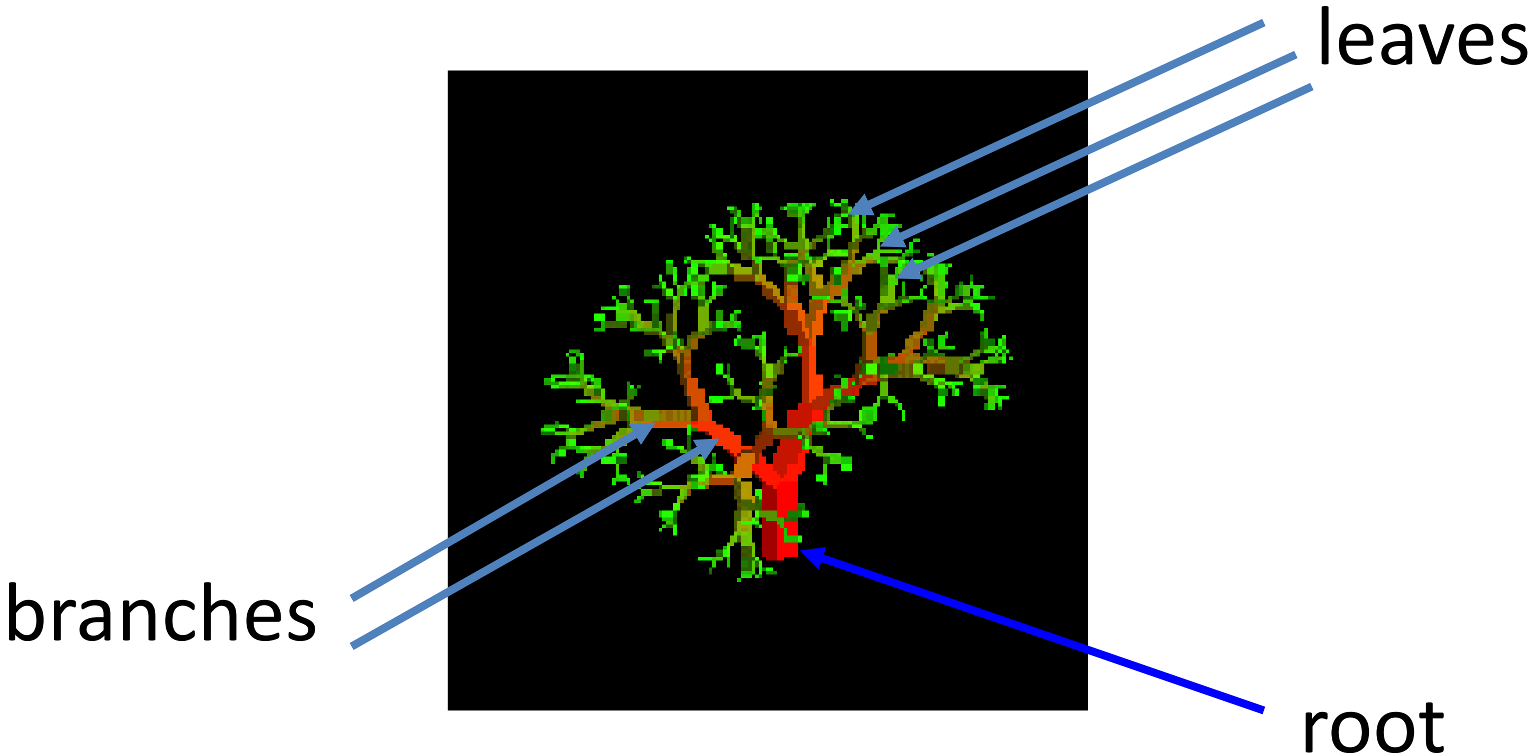
Hossein Rahmani

h_rahmani@iust.ac.ir

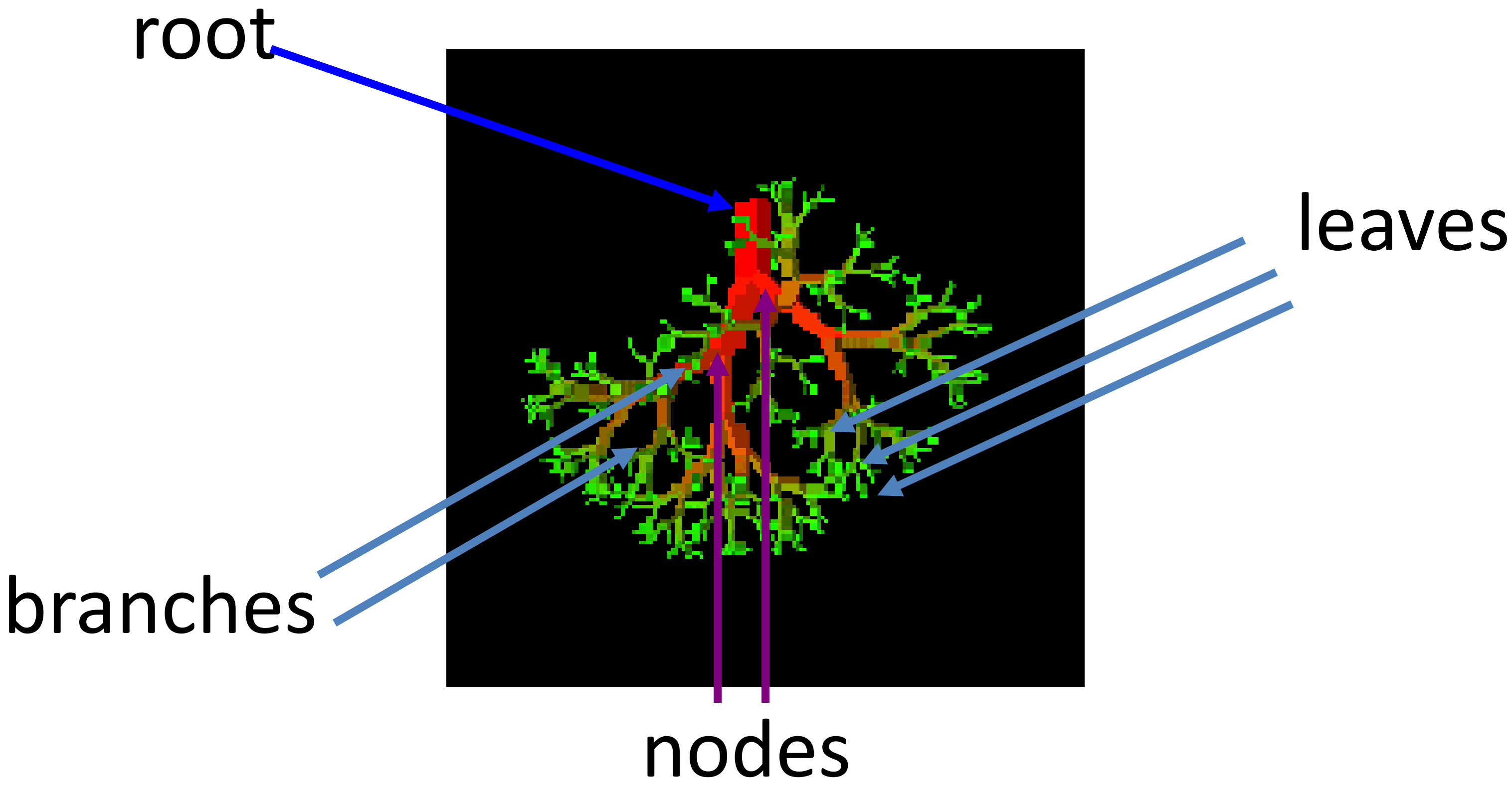
http://webpages.iust.ac.ir/h_rahmani/


Intro
Complexity
Data Structure
Trees
Hash Functions
Sorting
Dynamic Programming
Greedy Algorithm
Misc Graph/Tree Algorithms

Nature View of a Tree

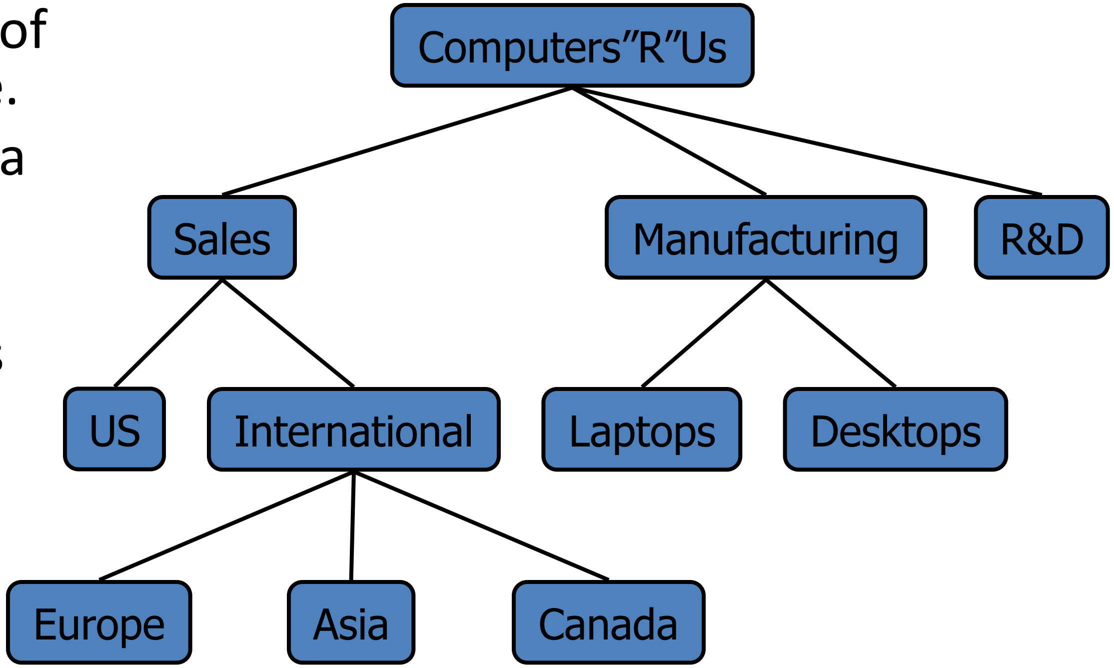


Computer Scientist's View



What is a Tree

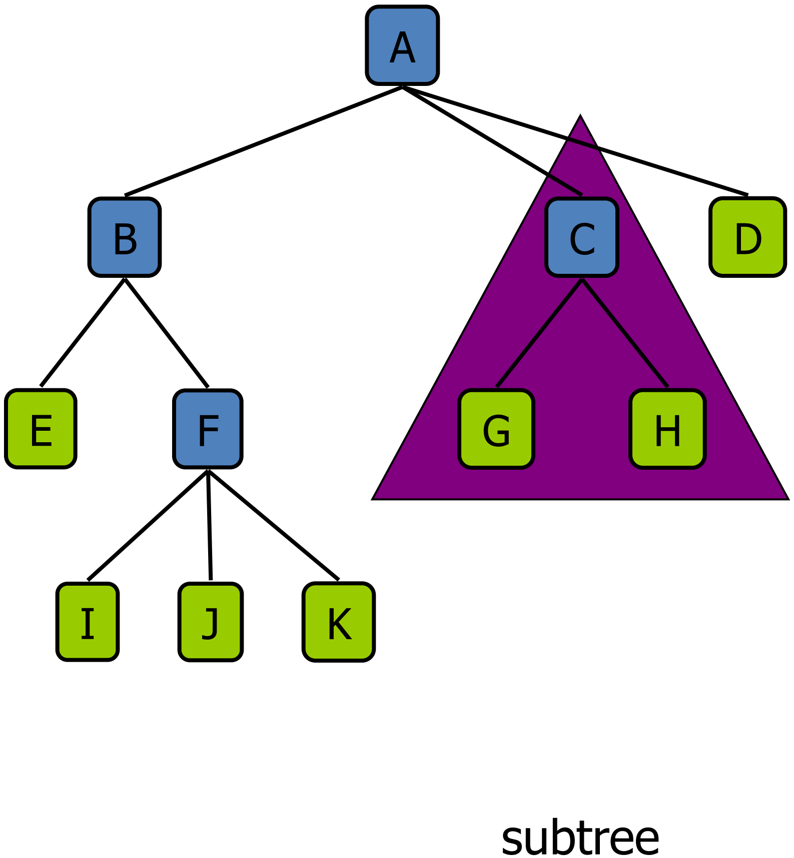
- A tree is a finite nonempty set of elements.
- It is an abstract model of a hierarchical structure.
- consists of nodes with a parent-child relation.
- Applications:
  - Organization charts
  - File systems
  - Programming environments



Tree Terminology

- **Root**: node without parent (A)
- **Siblings**: nodes share the same parent
- **Internal node**: node with at least one child (A, B, C, F)
- **External node** (leaf): node without children (E, I, J, K, G, H, D)
- **Ancestors** of a node: parent, grandparent, grand-grandparent, etc.
- **Descendant** of a node: child, grandchild, grand-grandchild, etc.
- **Depth** of a node: number of ancestors
- **Height** of a tree: maximum depth of any node (3)
- **Degree** of a node: the number of its children
- **Degree** of a tree: the maximum number of its node.

- **Subtree**: tree consisting of a node and its descendants



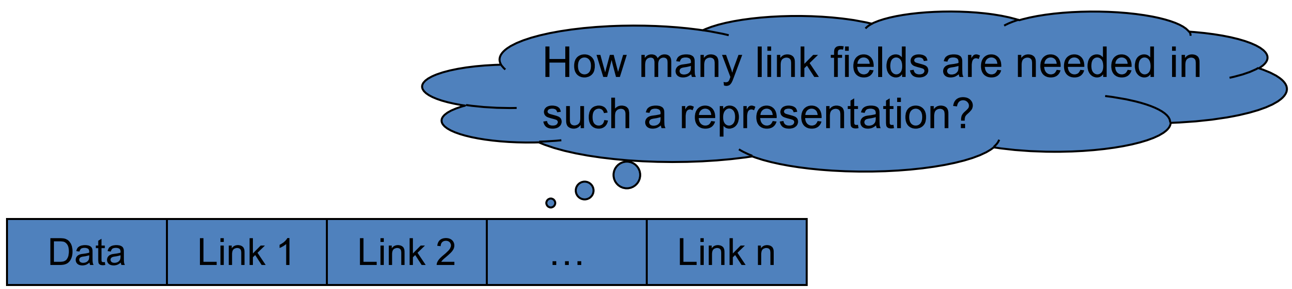
Tree ADT

- We use positions to abstract nodes
- Generic methods:
  - integer **size**()
  - boolean **isEmpty**()
  - objectIterator **elements**()
  - positionIterator **positions**()
- Accessor methods:
  - position **root**()
  - position **parent**(p)
  - positionIterator **children**(p)

- Query methods:
  - boolean **isInternal**(p)
  - boolean **isExternal**(p)
  - boolean **isRoot**(p)
- Update methods:
  - **swapElements**(p, q)
  - object **replaceElement**(p, o)
- Additional update methods may be defined by data structures implementing the Tree ADT

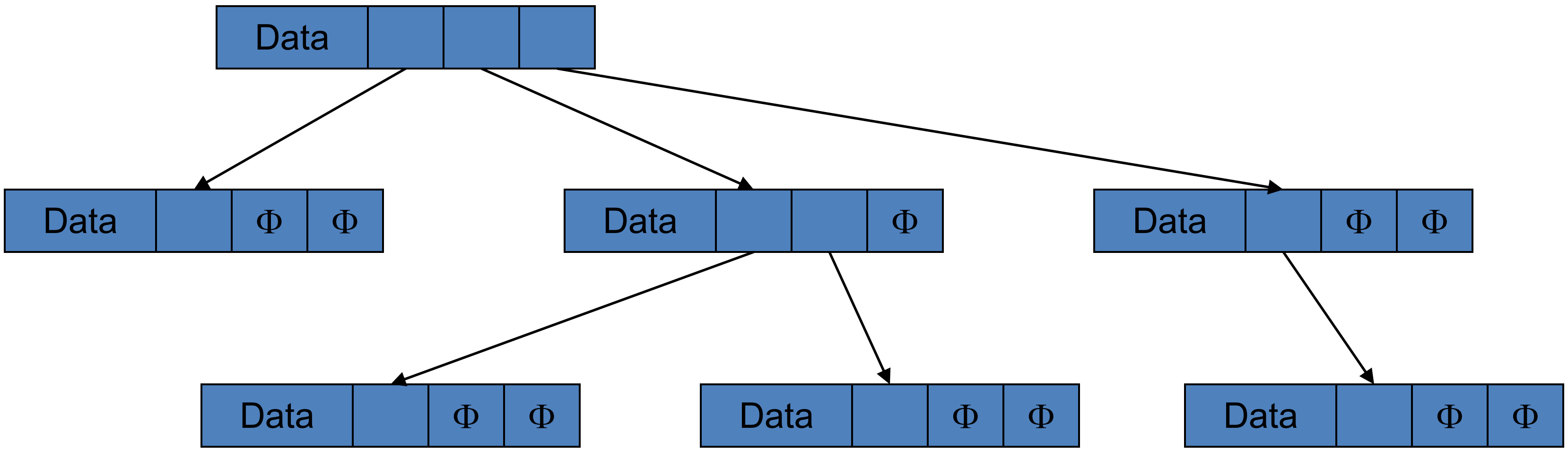
Intuitive Representation of Tree Node

- List Representation
  - (A (B (E (K, L), F), C (G), D (H (M), I, J)))
  - The root comes first, followed by a list of links to sub-trees



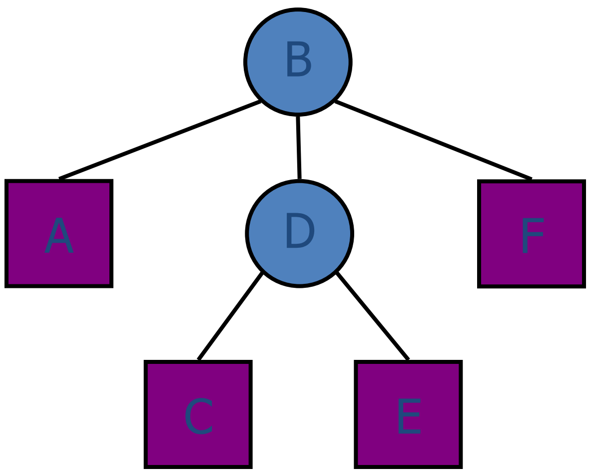
Trees

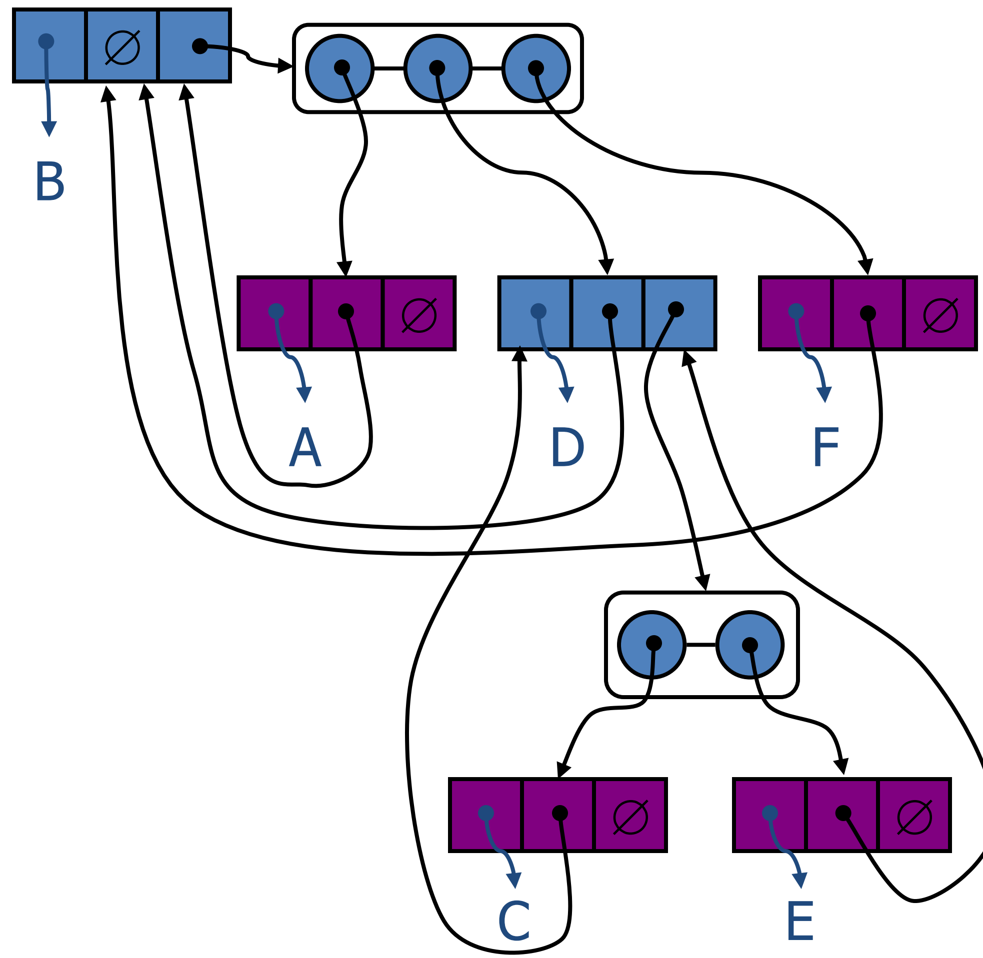
- Every tree node:
  - object – useful information
  - children – pointers to its children



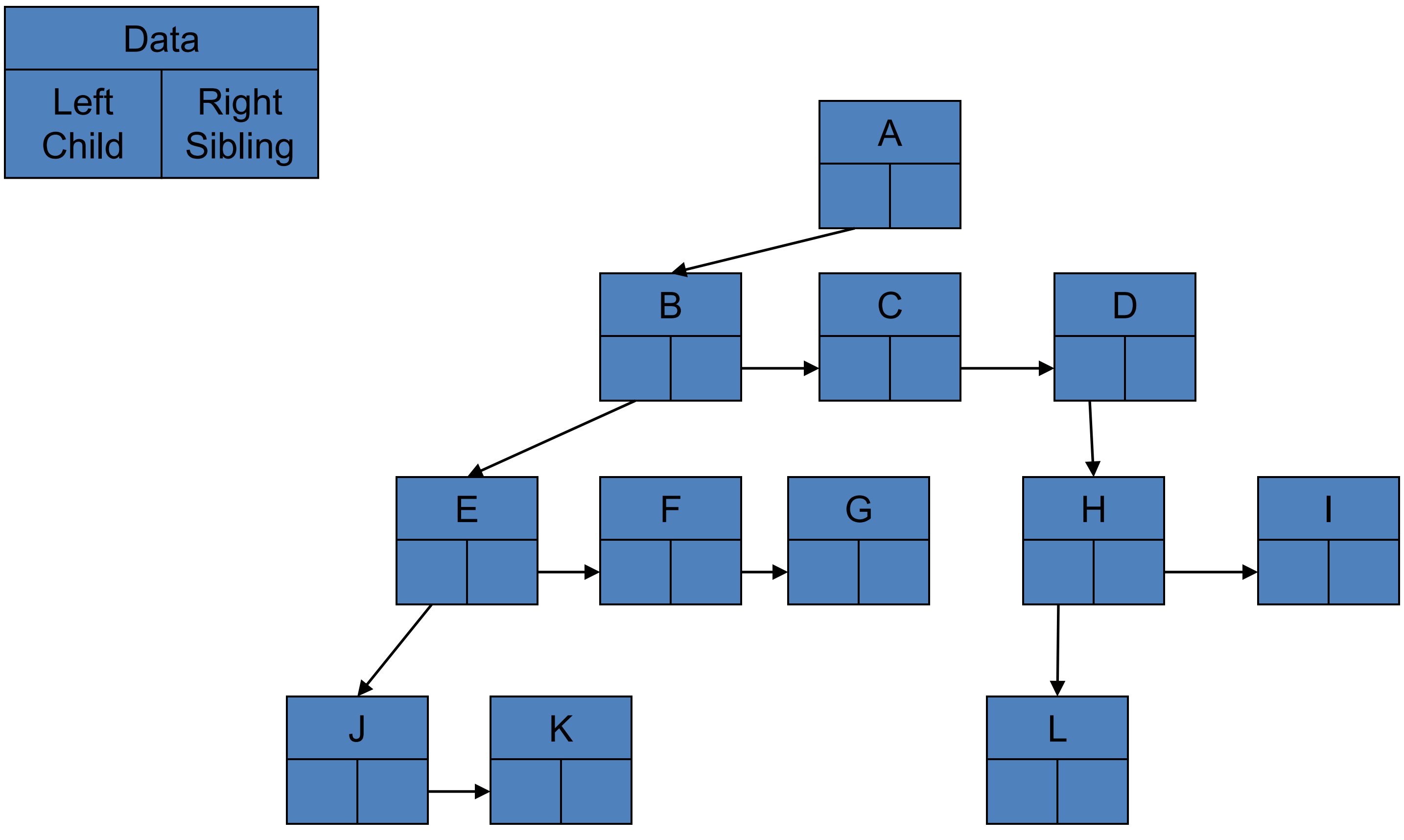
A Tree Representation

- A node is represented by an object storing
  - Element
  - Parent node
  - Sequence of children nodes





Left Child, Right Sibling Representation



Tree Traversal

- Two main methods:
  - Preorder
  - Postorder
- Recursive definition

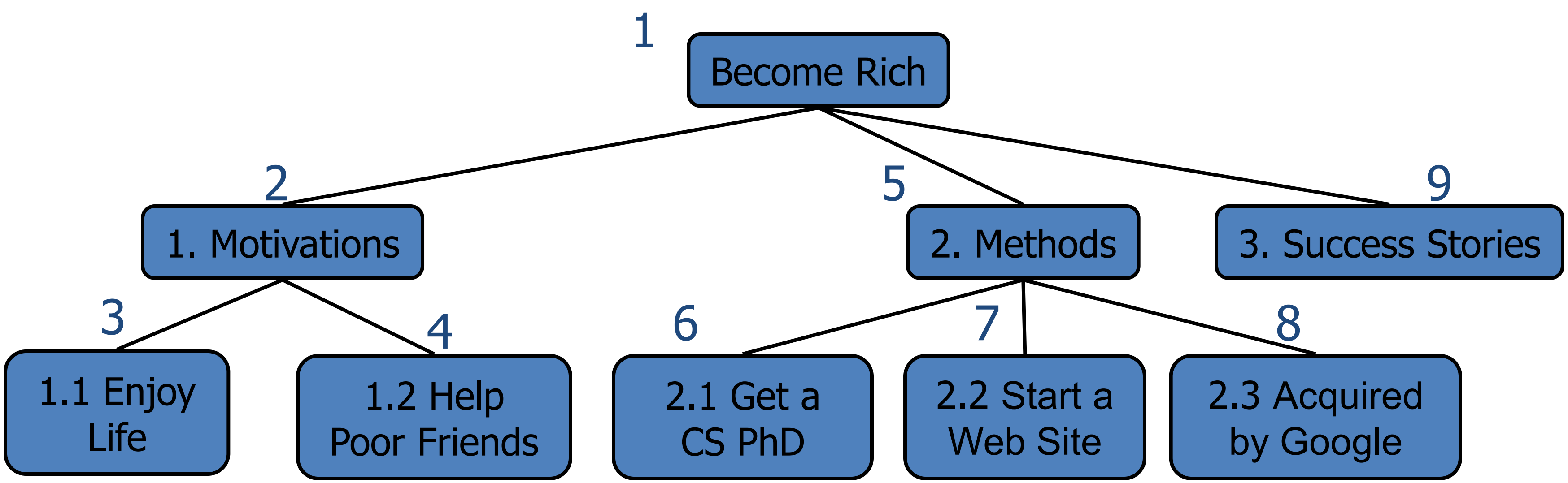
- Preorder:
  - visit the root
  - traverse in preorder the children (subtrees)

- Postorder
  - traverse in postorder the children (subtrees)
  - visit the root

Preorder Traversal

- A traversal visits the nodes of a tree in a systematic manner
- In a preorder traversal, a node is visited before its descendants
- Application: print a structured document

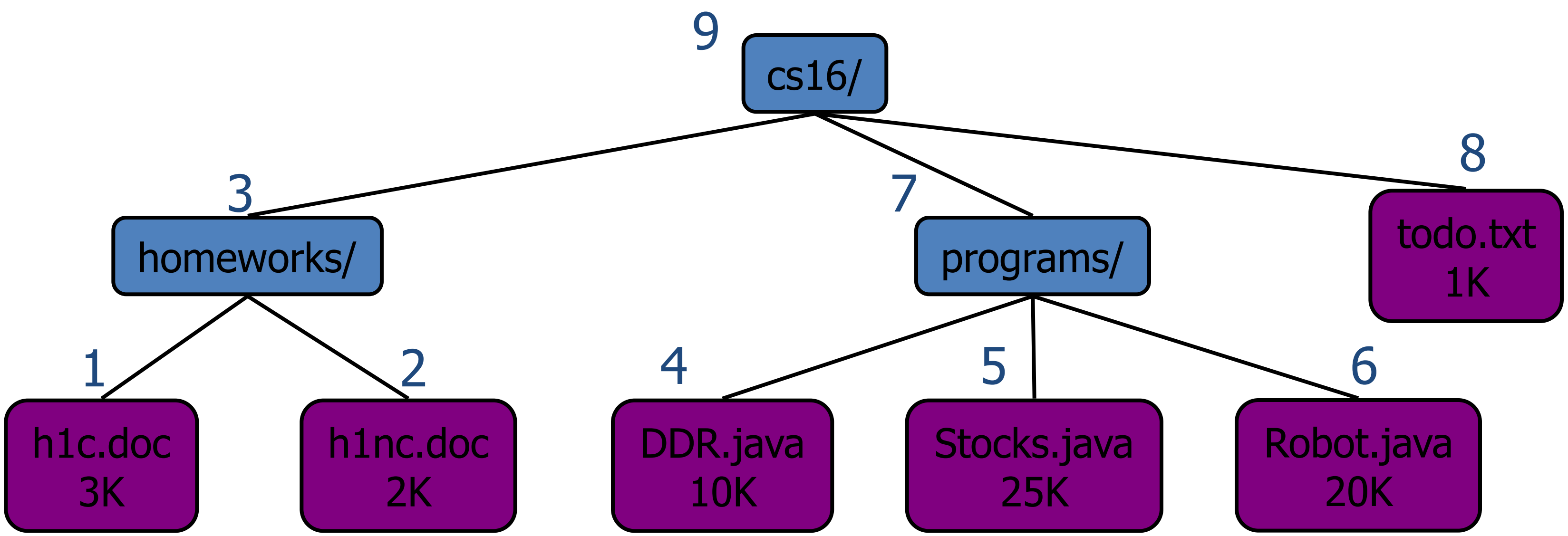
Algorithm ***preOrder***(***v***)
    visit(***v***)
    for each child ***w*** of ***v***
        preorder (***w***)

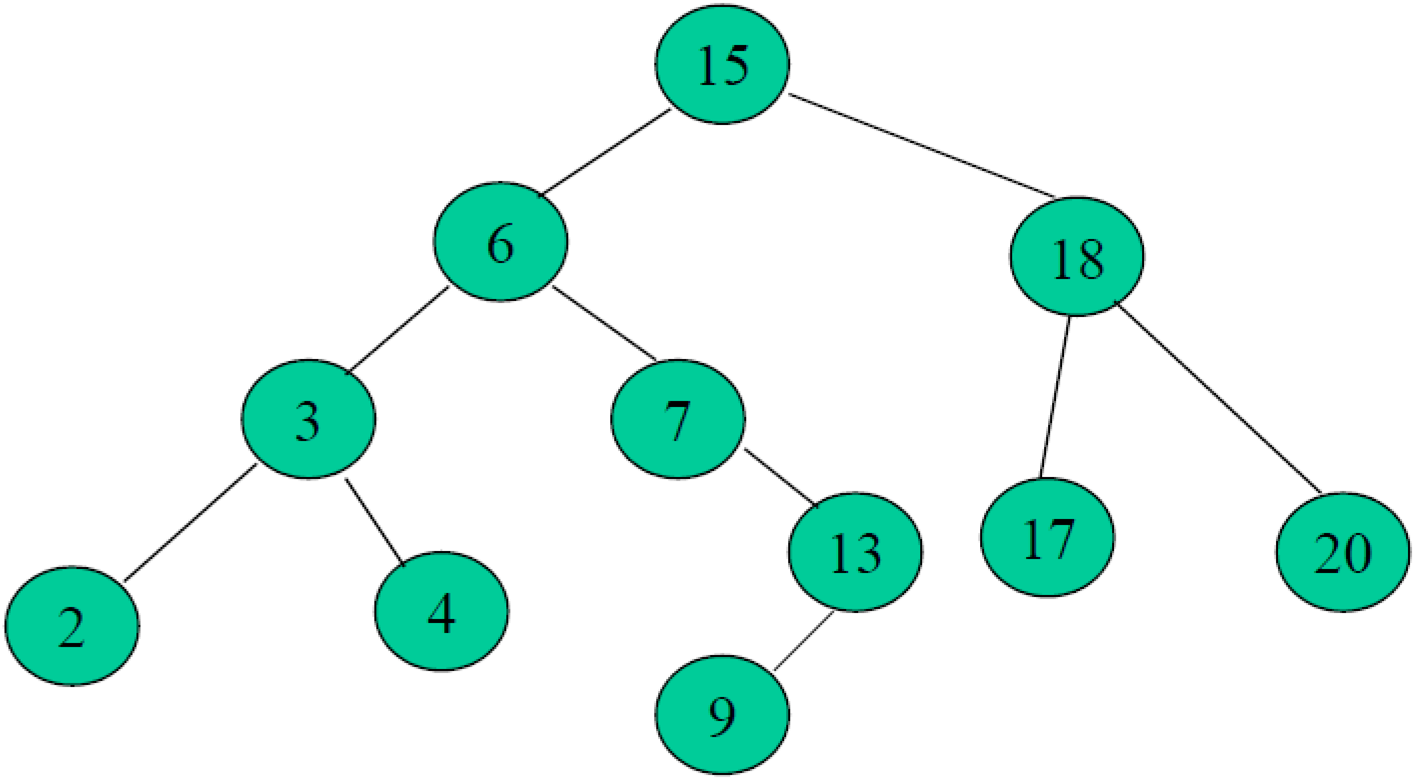


Postorder Traversal

- In a postorder traversal, a node is visited after its descendants
- Application: compute space used by files in a directory and its subdirectories

Algorithm ***postOrder***(***v***)
    for each child ***w*** of ***v***
        postOrder (***w***)
    visit(***v***)



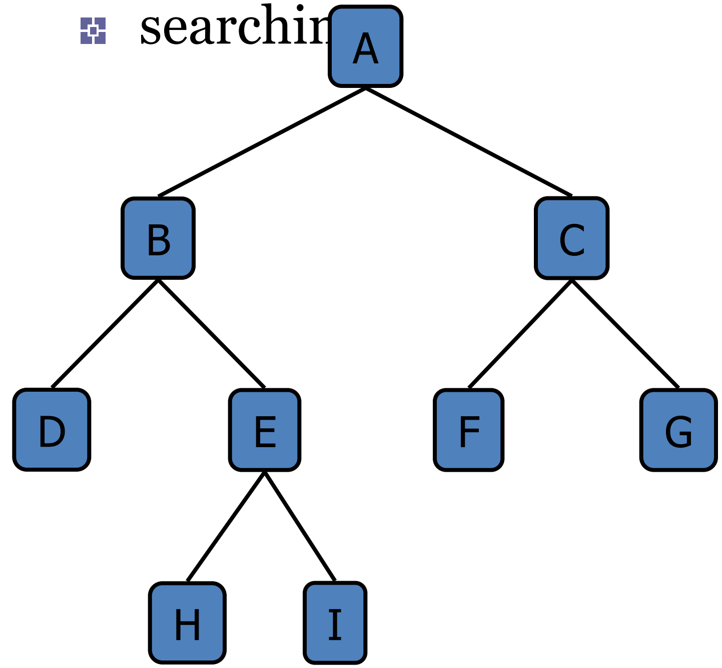


Preorder: 15, 6, 3, 2, 4, 7, 13, 9, 18, 17, 20
Inorder: 2, 3, 4, 6, 7, 9, 13, 15, 17, 18, 20
Postorder: 2, 4, 3, 9, 13, 7, 6, 17, 20, 18, 15

Binary Tree

- A binary tree is a tree with the following properties:
  - Each internal node has at **most** two children (degree of two)
  - The children of a node are an ordered pair
- We call the children of an internal node left child and right child
- Alternative recursive definition: a binary tree is either
  - a tree consisting of a single node, OR
  - a tree whose root has an ordered pair of children, each of which is a binary tree

- Applications:
  - arithmetic expressions
  - decision processes
  - searching

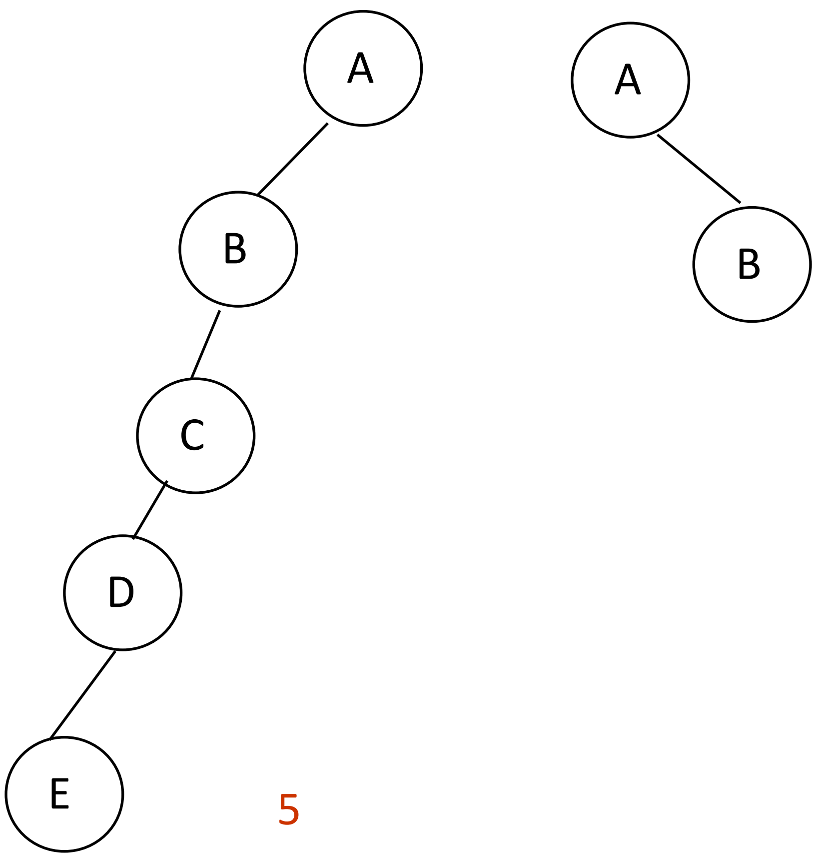


BinaryTree ADT

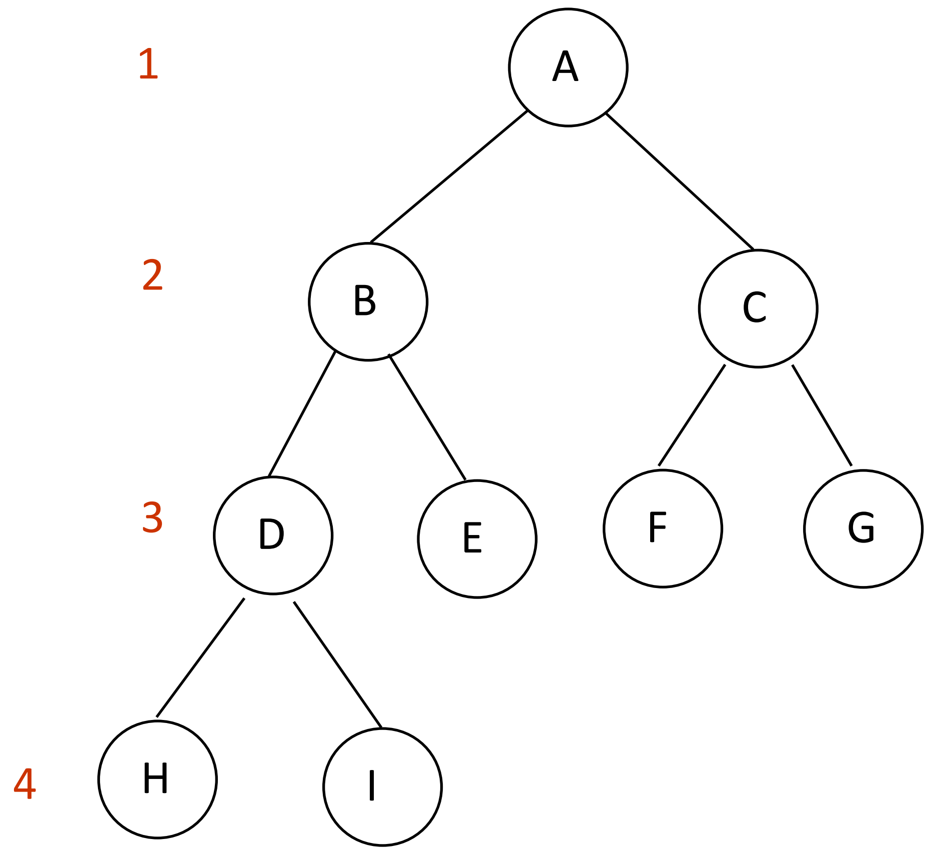
- The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT
- Additional methods:
  - position leftChild(p)
  - position rightChild(p)
  - position sibling(p)
- Update methods may be defined by data structures implementing the BinaryTree ADT

Examples of the Binary Tree

Skewed Binary Tree



Complete Binary Tree



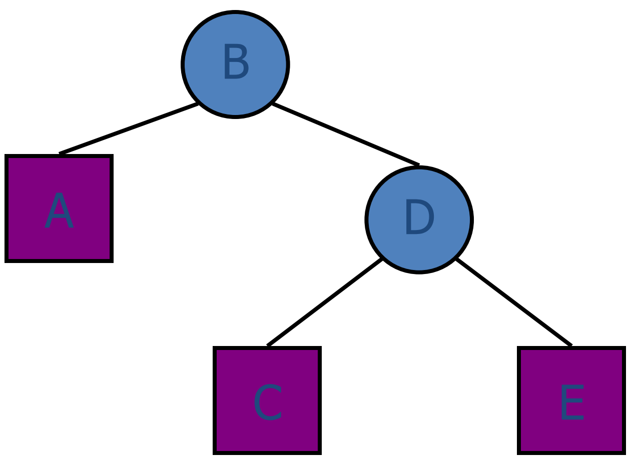
Differences Between A Tree and A Binary Tree

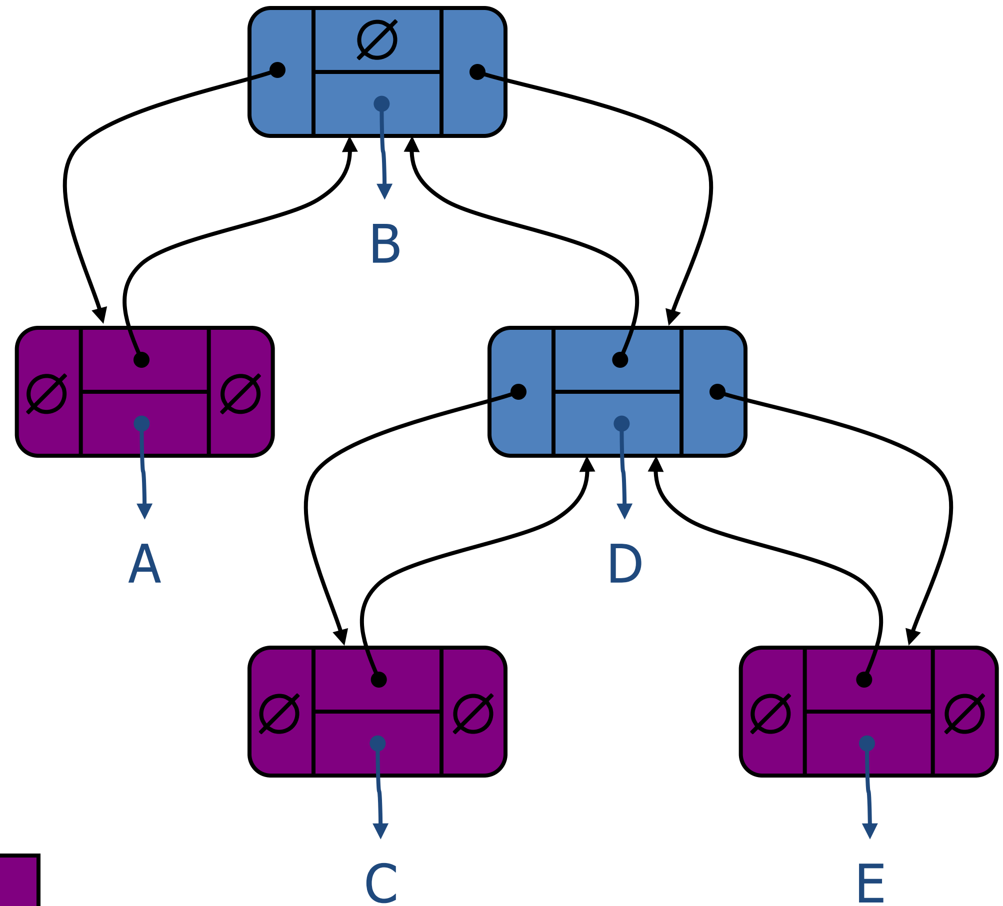
- The subtrees of a binary tree are ordered; those of a tree are not ordered.

- Are different when viewed as binary trees.
- Are the same when viewed as trees.

Data Structure for Binary Trees

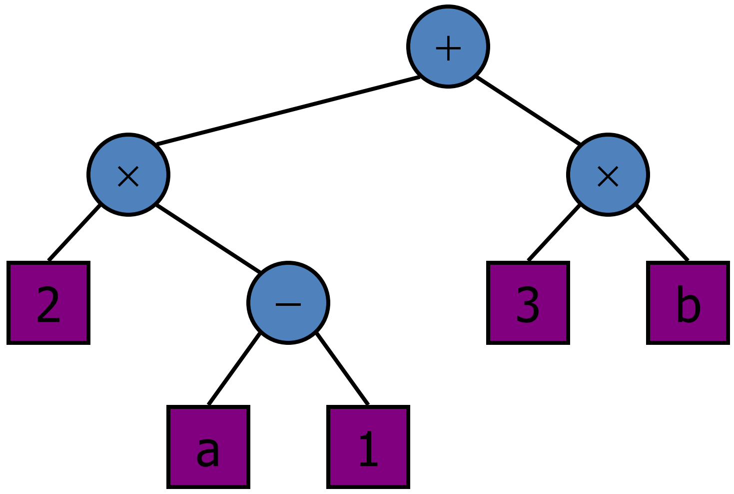
- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node





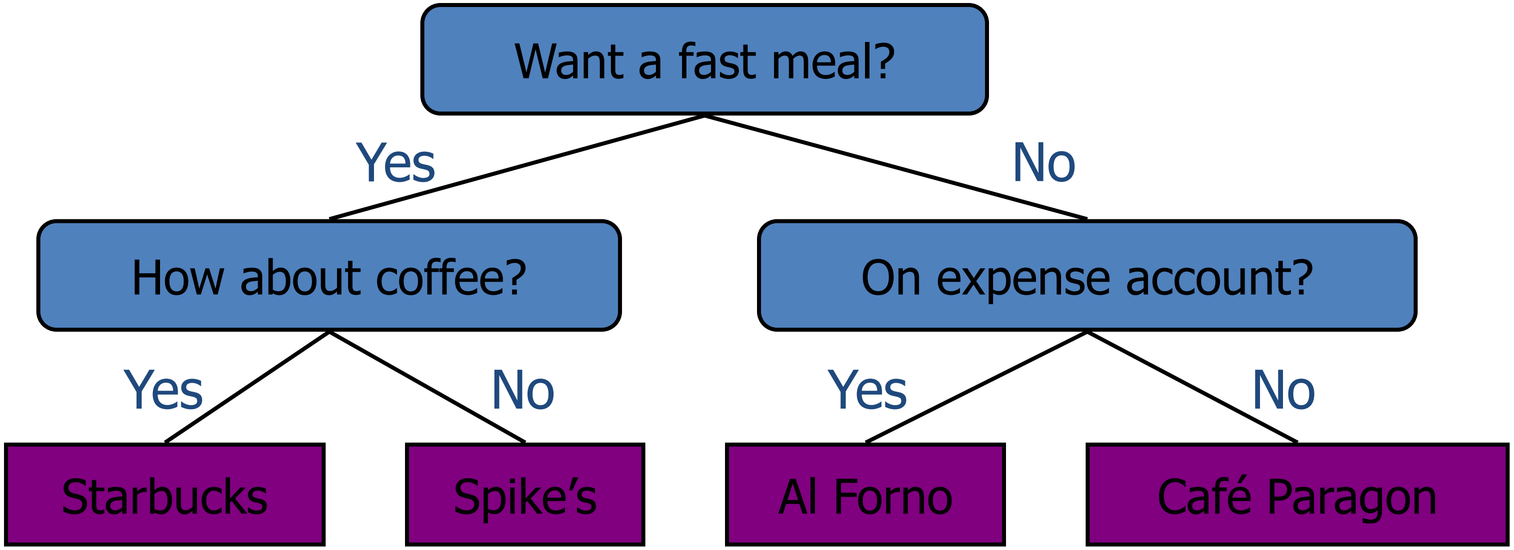
Arithmetic Expression Tree

- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands
- Example: arithmetic expression tree for the expression $(2 \times (a - 1) + (3 \times b))$



Decision Tree

- Binary tree associated with a decision process
  - <u>internal</u> nodes: questions with yes/no answer
  - <u>external</u> nodes: decisions
- Example: dining decision



Maximum Number of Nodes in a Binary Tree

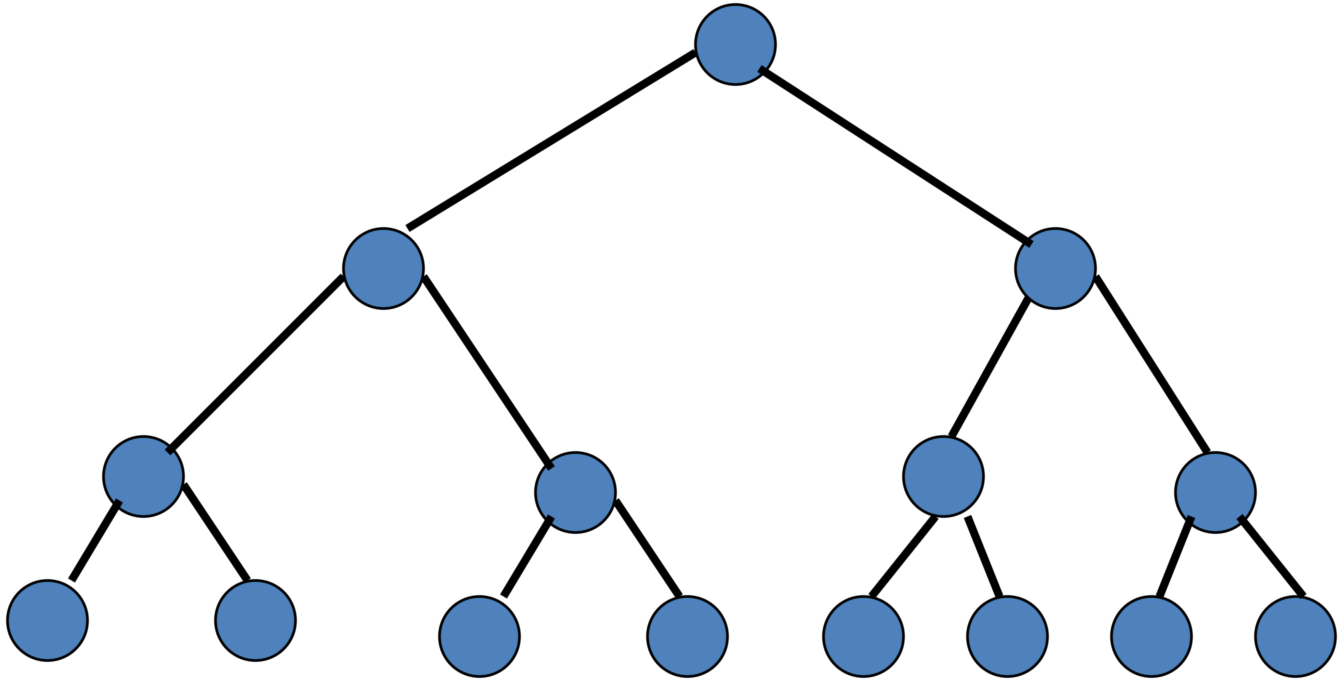
- The maximum number of nodes on depth i of a binary tree is 2^i, i>=0.
- The maximum nubmer of nodes in a binary tree of height k is $2^{k+1}-1$, k>=0.

Prove by induction.

$$\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$$

Full Binary Tree

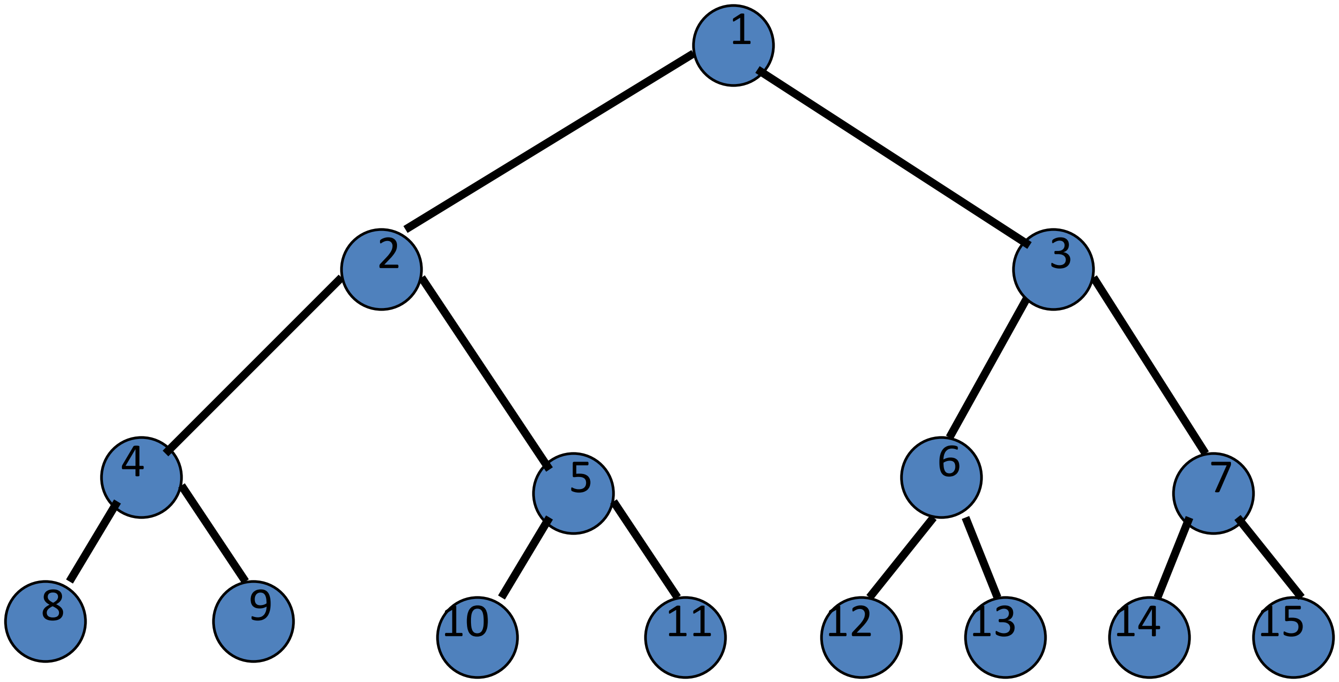
- A full binary tree of a given height k has $2^{k+1}-1$ nodes.



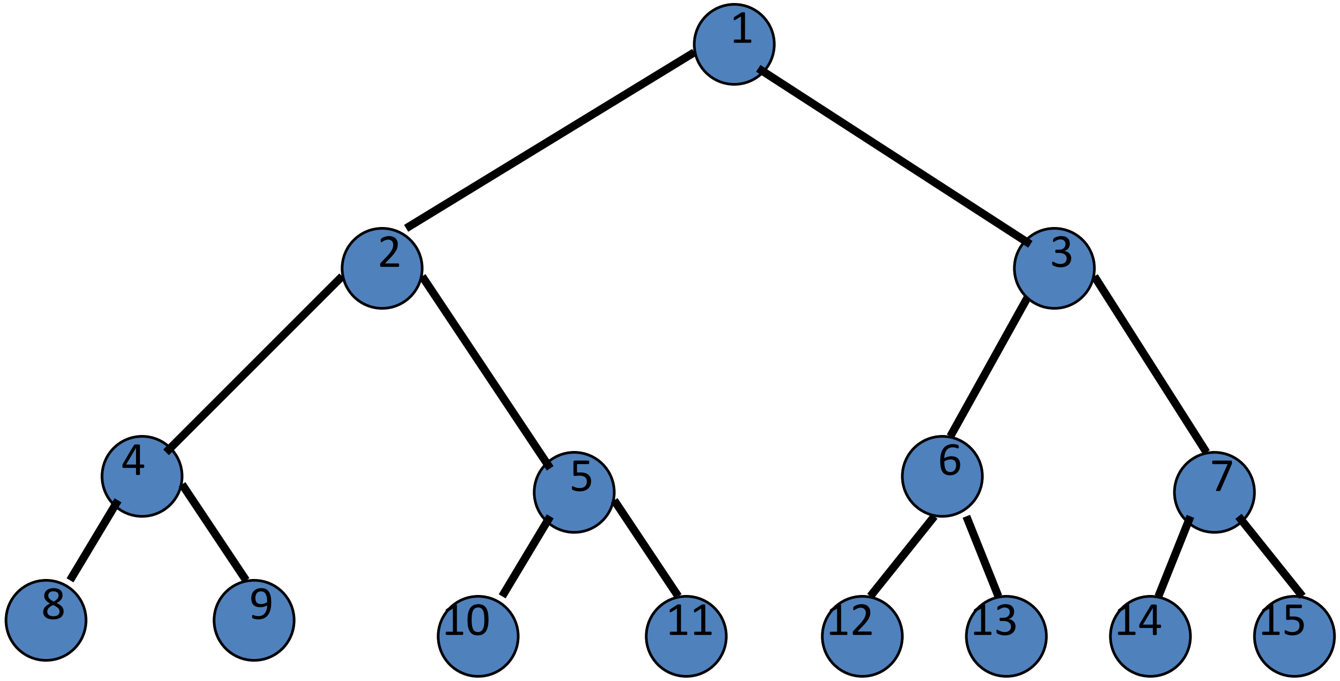
Height 3 full binary tree.

Labeling Nodes In A Full Binary Tree

- Label the nodes 1 through $2^{k+1} - 1$.
- Label by levels from top to bottom.
- Within a level, label from left to right.

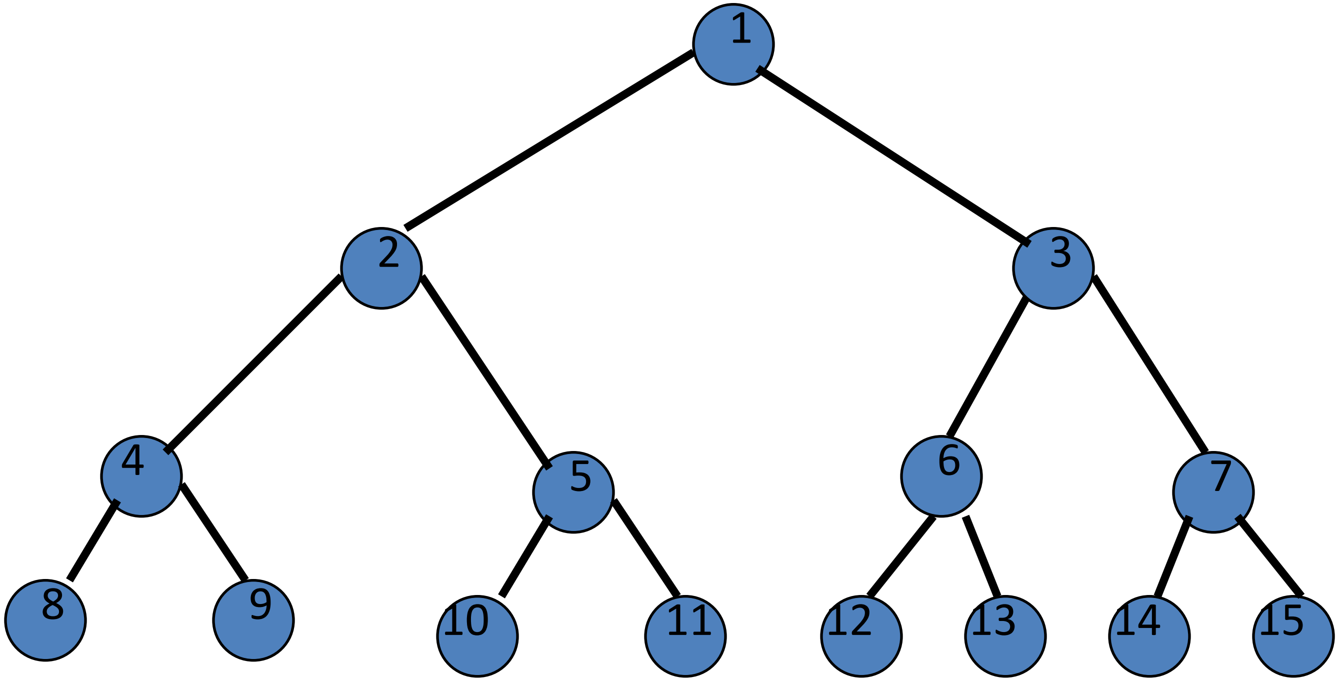


Node Number Properties



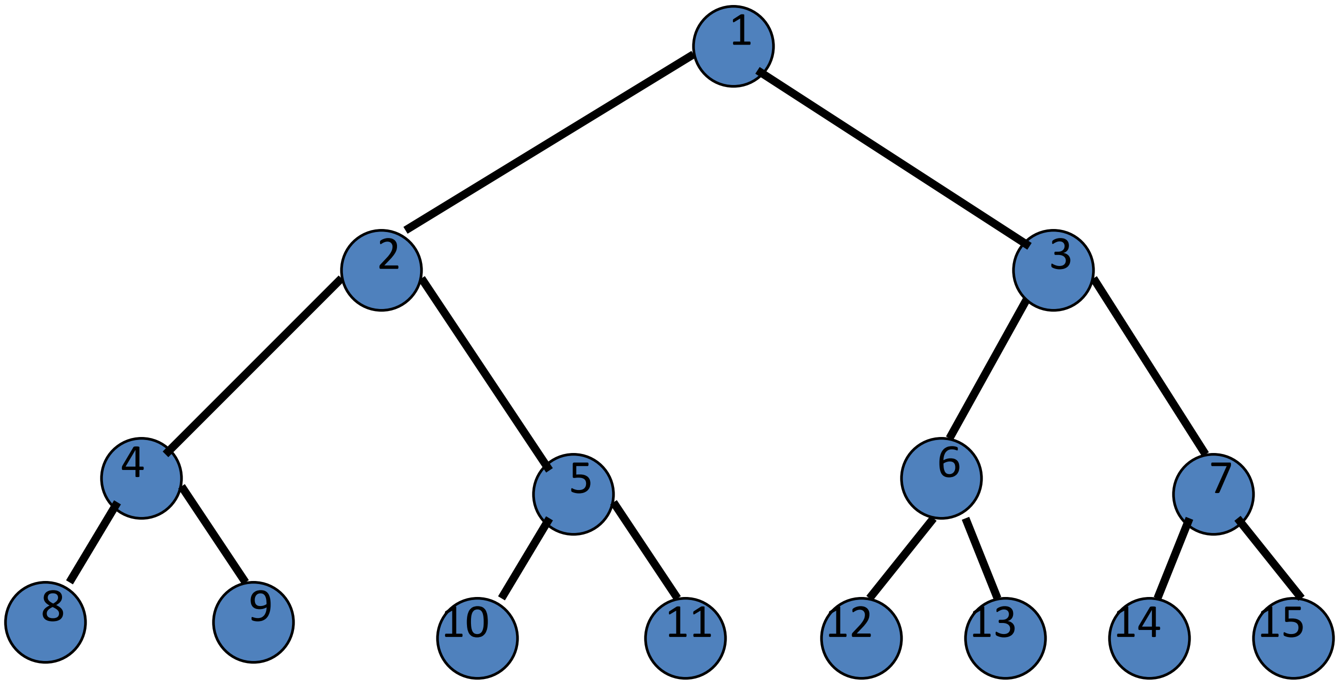
- Parent of node i is node i / 2, unless i = 1.
- Node 1 is the root and has no parent.

Node Number Properties



- Left child of node i is node $2i$, unless $2i > n$, where n is the number of nodes.
- If $2i > n$, node i has no left child.

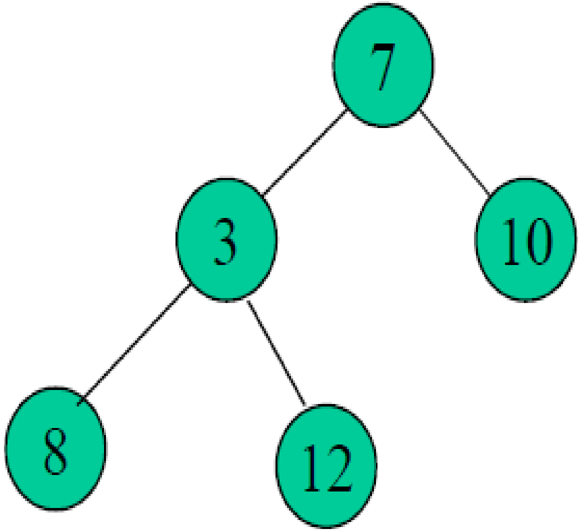
Node Number Properties



- Right child of node i is node $2i+1$, unless $2i+1 > n$, where n is the number of nodes.
- If $2i+1 > n$, node i has no right child.

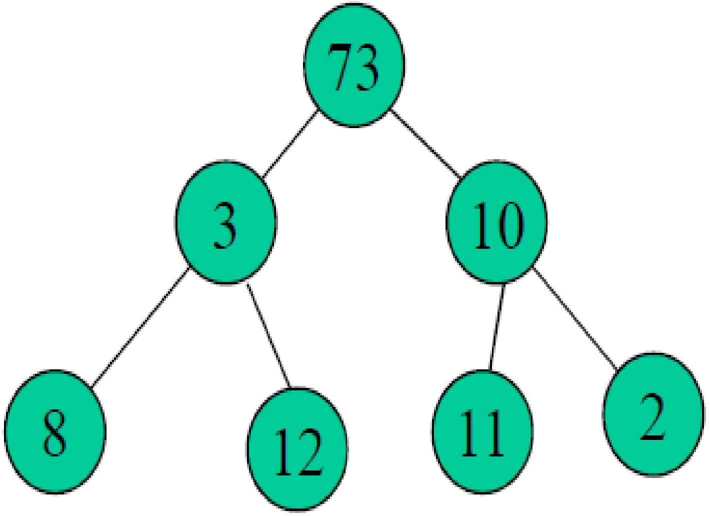
Full binary tree:

Each node is either a leaf or has degree exactly 2.



Complete binary tree:

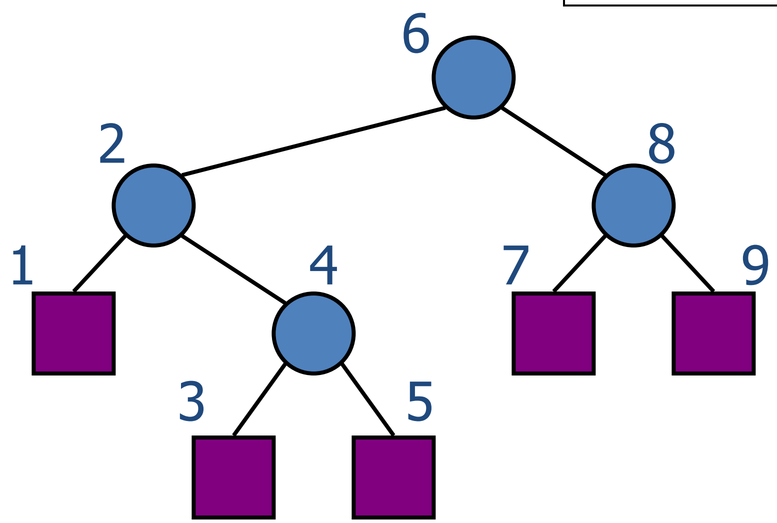
All leaves have the same depth and all internal nodes have degree 2.



(Binary Tree Traversals)
Inorder Traversal

- In an inorder traversal a node is visited after its left subtree and before its right subtree

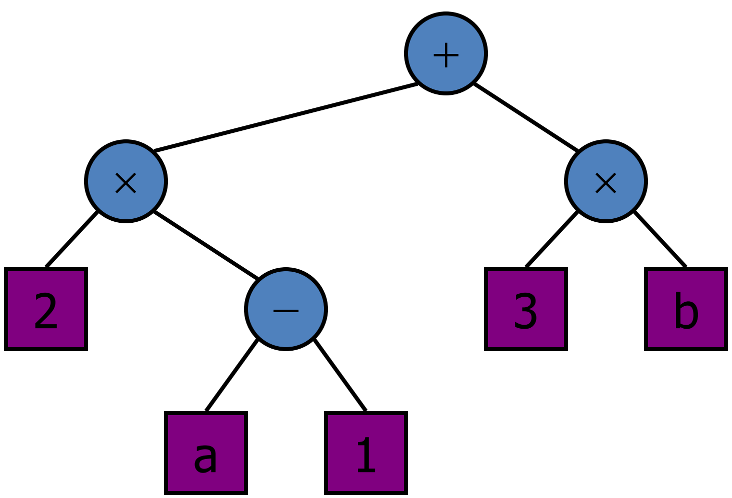
Algorithm ***inOrder***(***v***)
    if ***isInternal*** (***v***)
        inOrder (***leftChild*** (***v***))
    visit(***v***)
    if ***isInternal*** (***v***)
        inOrder (***rightChild*** (***v***))



Print Arithmetic Expressions

- Specialization of an inorder traversal
  - print operand or operator when visiting node
  - print “(“ before traversing left subtree
  - print “)“ after traversing right subtree

```
Algorithm inOrder (v)
    if isInternal (v){
        print(“(”)
        inOrder (leftChild (v))}
    print(v.element ())
    if isInternal (v){
        inOrder (rightChild (v))
        print (“)”)}
```

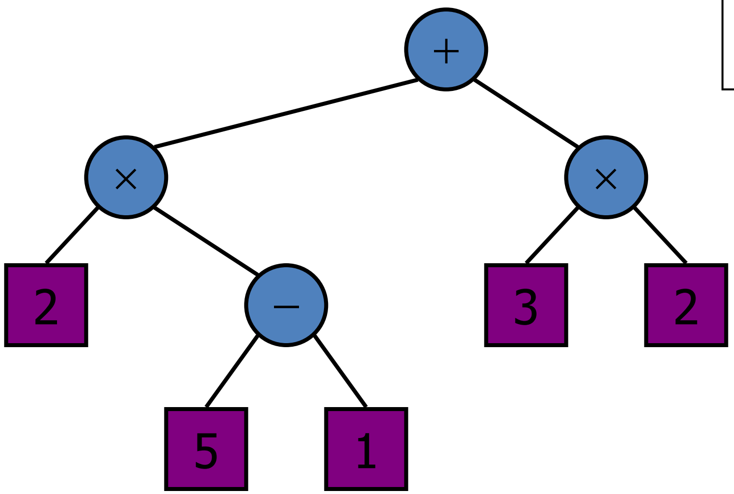


$((2 \times (a - 1)) + (3 \times b))$

Evaluate Arithmetic Expressions

- recursive method returning the value of a subtree
- when visiting an internal node, combine the values of the subtrees

Algorithm ***evalExpr***(***v***)
    if ***isExternal*** (***v***)
        return ***v.element*** ()
    else
        x $\leftarrow$ ***evalExpr***(***leftChild*** (***v***))
        y $\leftarrow$ ***evalExpr***(***rightChild*** (***v***))
        ◊ $\leftarrow$ operator stored at ***v***
        return ***x*** ◊ ***y***



Quiz 1

- In a complete k-ary tree, every internal node has exactly k children or no child. The number of leaves in such a tree with n internal nodes is:

A	nk
B	(n – 1) k+ 1
C	n(k – 1) + 1
D	n(k – 1)

Quiz 2

- The inorder and preorder traversal of a binary tree are:

  d b e a f c g and

  a b d e c f g, respectively.

  The postorder traversal of the binary tree is?

Quiz 3

- What does the following function do for a given binary tree?

```
int fun(struct node *root)
{
 if (root == NULL) return 0;
 if (root->left == NULL && root->right == NULL)
    return 0;
 return 1 + fun(root->left) + fun(root->right);
}
```