MECHTRON 2MD3

Data Structures and Algorithms for Mechatronics

Winter 2022

24 Binary Trees

Department of Computing and Software

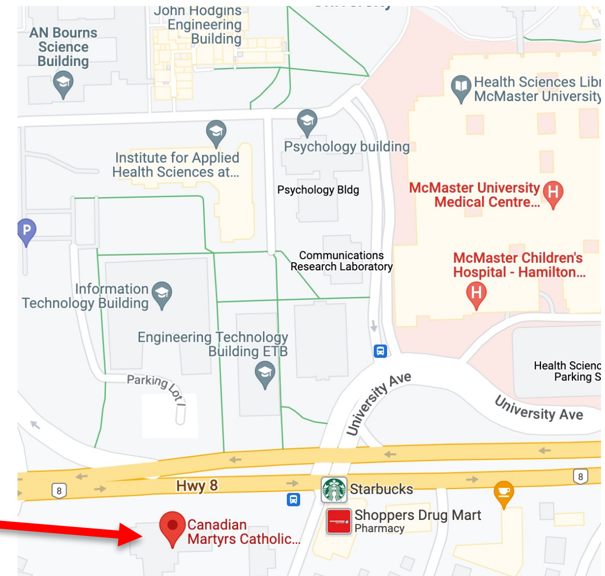
Instructor:

Omid Isfahanialamdari

March 17, 2022

Admin

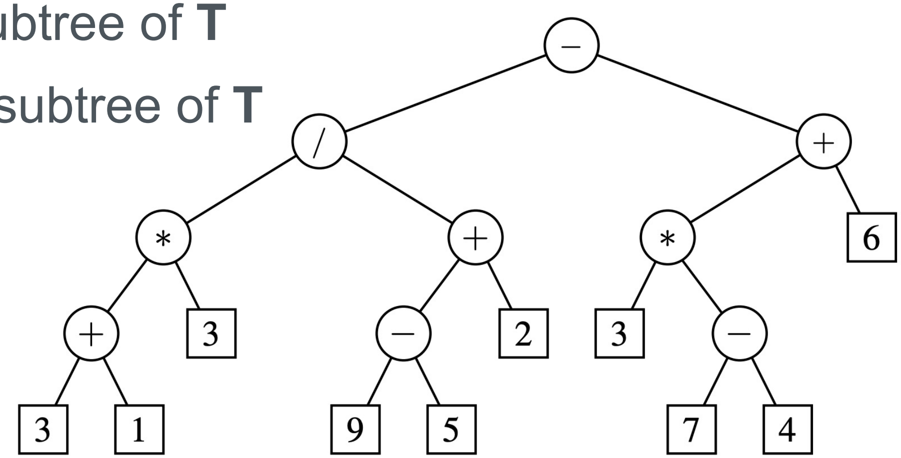
- Mid-Term 2:
  - Wednesday 23 March 2022
  - Duration: **1 hour**
  - **From 1:30 to 14:30 (lec. time)**
  - Location: **MCMST CDN_MARTYRS**
    - Seems to be here, I am not sure



  - Covers: Topics from “Doubly Linked Lists” until the lecture of Wednesday 16 March 2022 (inclusive)

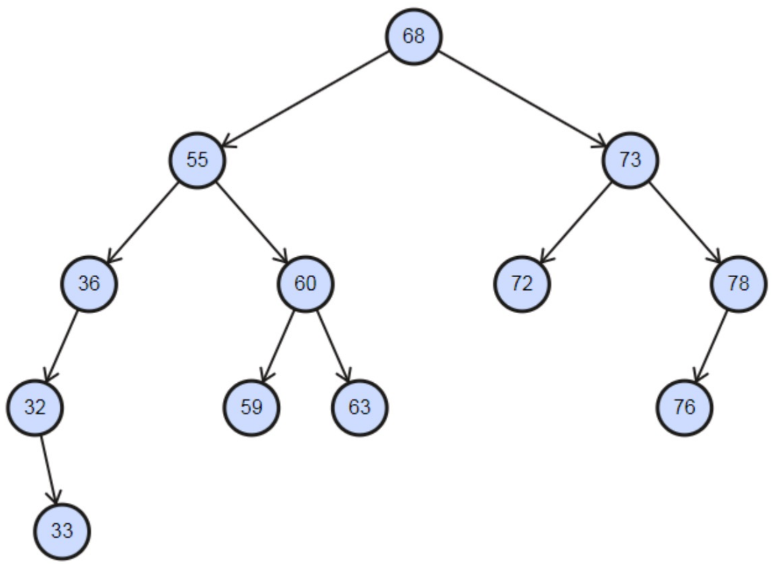
Binary Tree

- A **binary tree** is an **ordered tree** with the following properties:
  - Every node has at most two children.
  - Each child node is labeled as a left child or a right child.
  - A left child **precedes** a right child in the **order** of children.
- A recursive definition of the binary tree:
  - A binary tree is either **empty** or consists of:
    - A node **r**, called the **root** of **T** and storing an element
    - A binary tree, called the **left** subtree of **T**
    - A binary tree, called the **right** subtree of **T**
- Applications:
  - arithmetic expressions
  - decision processes
  - searching



Binary Tree - Some Definitions

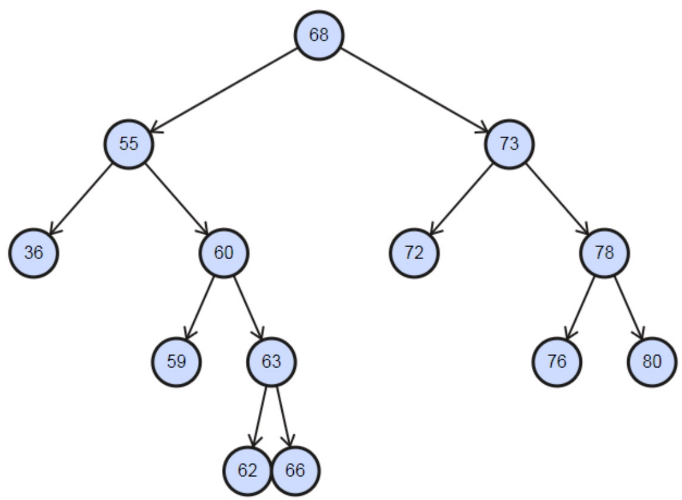
- Left subtree: Subtree rooted at the left child of an internal node
- Right subtree: Subtree rooted at the right child of an internal node
- Proper/full tree: A tree in which every node has either 0 or 2 children
- Complete tree: Tree in which all except possibly the last level is completely filled and the nodes in the last level are as far left as possible
- Perfect tree: Complete tree in which the last level is completely filled



Binary ✓, Proper ✗, Complete ✗, Perfect ✗

Binary Tree - Some Definitions

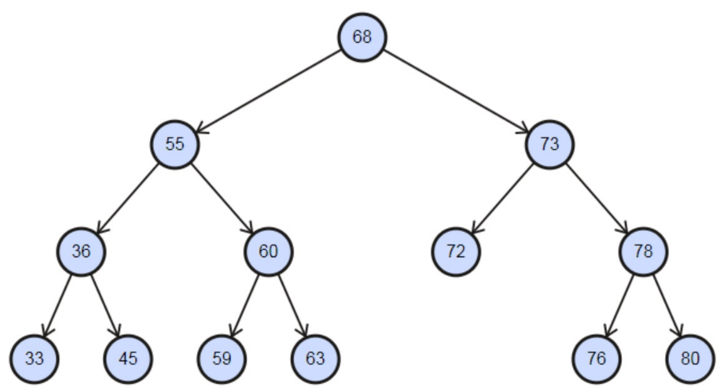
- Left subtree: Subtree rooted at the left child of an internal node
- Right subtree: Subtree rooted at the right child of an internal node
- Proper/full tree: A tree in which every node has either 0 or 2 children
- Complete tree: Tree in which all except possibly the last level is completely filled and the nodes in the last level are as far left as possible
- Perfect tree: Complete tree in which the last level is completely filled



Binary ✓, Proper ✓, Complete ✗, Perfect ✗

Binary Tree - Some Definitions

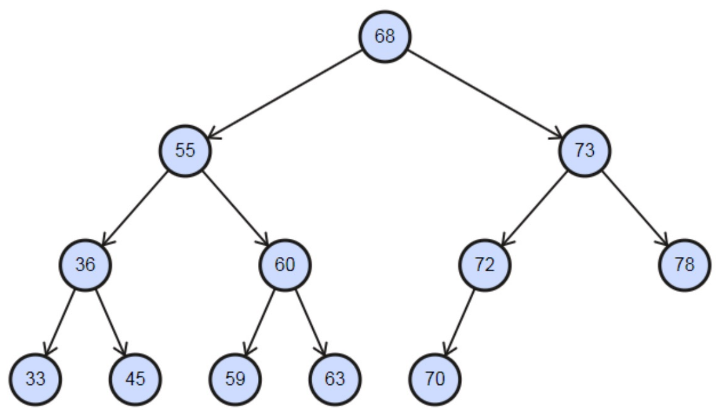
- Left subtree: Subtree rooted at the left child of an internal node
- Right subtree: Subtree rooted at the right child of an internal node
- Proper/full tree: A tree in which every node has either 0 or 2 children
- Complete tree: Tree in which all except possibly the last level is completely filled and the nodes in the last level are as far left as possible
- Perfect tree: Complete tree in which the last level is completely filled



Binary ✓, Proper ✓, Complete ✗, Perfect ✗

Binary Tree - Some Definitions

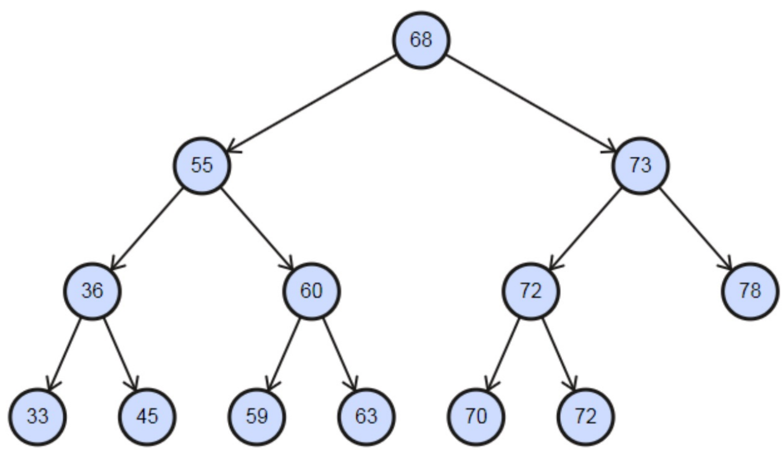
- Left subtree: Subtree rooted at the left child of an internal node
- Right subtree: Subtree rooted at the right child of an internal node
- Proper/full tree: A tree in which every node has either 0 or 2 children
- Complete tree: Tree in which all except possibly the last level is completely filled and the nodes in the last level are as far left as possible
- Perfect tree: Complete tree in which the last level is completely filled



Binary ✓, Proper ✗, Complete ✓, Perfect ✗

Binary Tree - Some Definitions

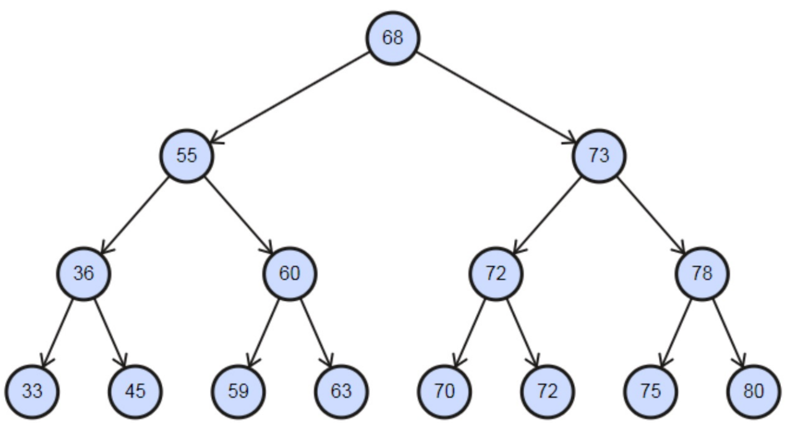
- Left subtree: Subtree rooted at the left child of an internal node
- Right subtree: Subtree rooted at the right child of an internal node
- Proper/full tree: A tree in which every node has either 0 or 2 children
- Complete tree: Tree in which all except possibly the last level is completely filled and the nodes in the last level are as far left as possible
- Perfect tree: Complete tree in which the last level is completely filled



Binary ✓, Proper ✓, Complete ✓, Perfect ✗

Binary Tree - Some Definitions

- Left subtree: Subtree rooted at the left child of an internal node
- Right subtree: Subtree rooted at the right child of an internal node
- Proper/full tree: A tree in which every node has either 0 or 2 children
- Complete tree: Tree in which all except possibly the last level is completely filled and the nodes in the last level are as far left as possible
- Perfect tree: Complete tree in which the last level is completely filled



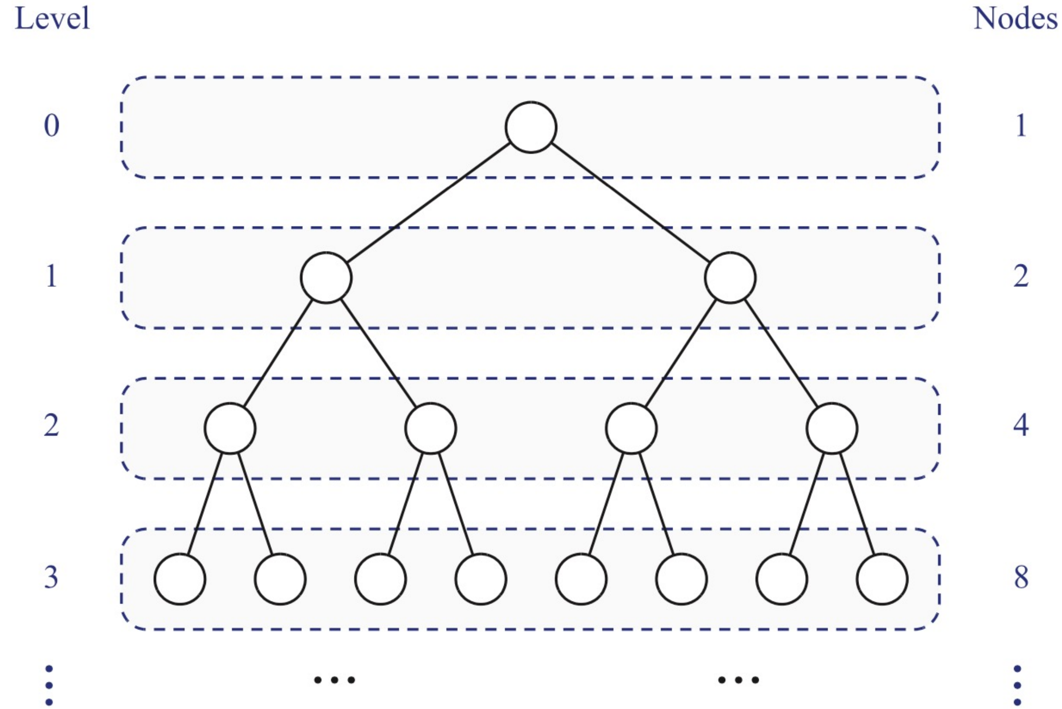
Binary ✓, Proper ✓, Complete ✓, Perfect ✓

Properties of Binary Trees

- Relationships between the height of a binary tree and the number of its nodes is interesting
- Recall: Height is the maximum depth of the tree, and depth of a node is the number of its ancestors.
- maximum number of

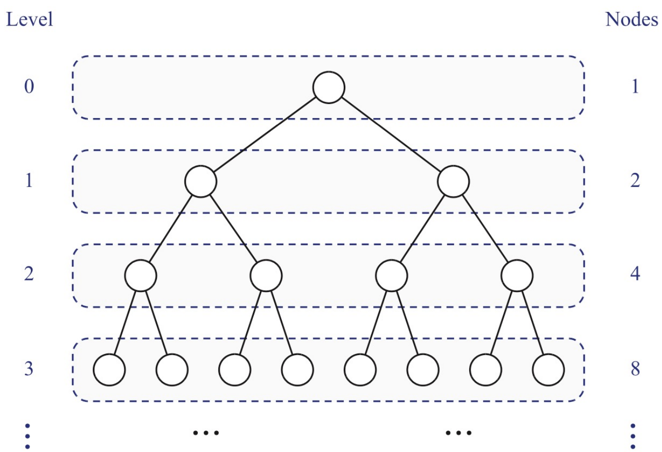
nodes in each level:

- n_E = # of external nodes
- n_I = # of internal nodes
- $n = n_E + n_I$
- h = height of the tree



Properties of Binary Trees

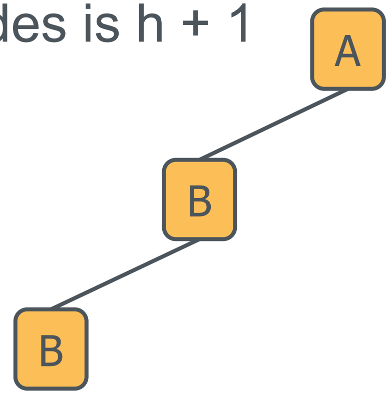
- maximum number of nodes at level **i** of a binary tree:
  - $\mathbf{2^i}$, for $i \geq 0$.

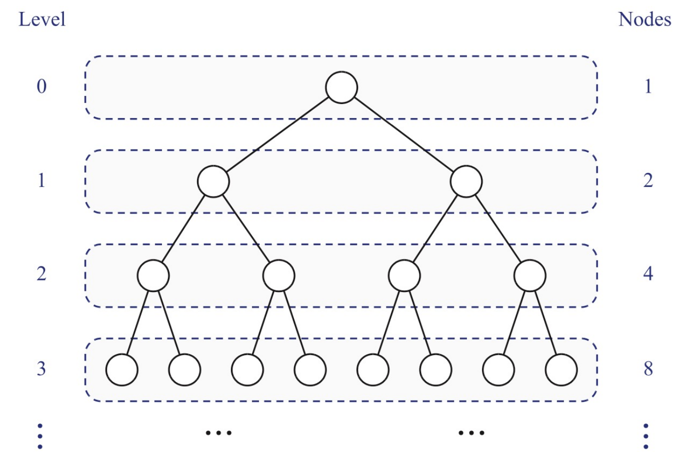


- Proof:
  - by Induction:
    - Introduction base:
      - i = 0 (level 0 that has only root):
        - The number of node is: $2^i = 2^0 = 1$.
    - Induction hypothesis
      - Assume that for $i \geq 0$, the maximum number of nodes on level **i-1** is $\mathbf{2^{i-1}}$.
    - Induction step:
    - Since each node in a binary tree has a maximum degree of 2, therefore, the maximum number of nodes on level **i** is $\mathbf{2 * 2^{i-1} = 2^i}$

Properties of Binary Trees

- maximum number of nodes at level **i** of a binary tree is $\mathbf{2^i}$, for $i \geq 0$.
- $h + 1 \leq n \leq 2^{h+1} - 1$
  - min # of nodes is $h + 1$

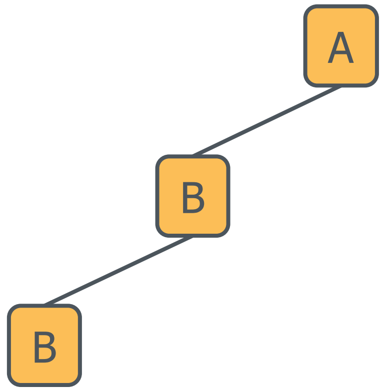


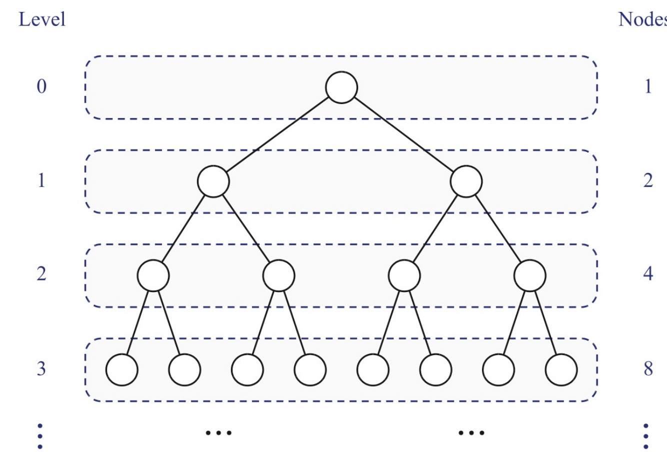


  - max # of nodes is:
    - $1 + 2 + 4 + 8 + \ldots + 2^h = 2^{h+1} - 1$
      - summation of geometrical series

Properties of Binary Trees

- maximum number of nodes at level **i** of a binary tree is $\mathbf{2^i}$, for $i \geq 0$.
- $h + 1 \leq n \leq 2^{h+1} - 1$
- $1 \leq n_E \leq 2^h$
- $h \leq n_I \leq 2^h - 1$
- $\log(n + 1) - 1 \leq h \leq n - 1$



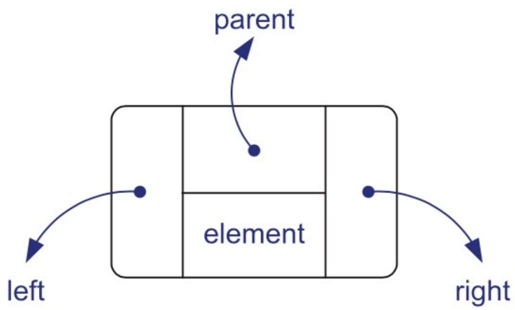


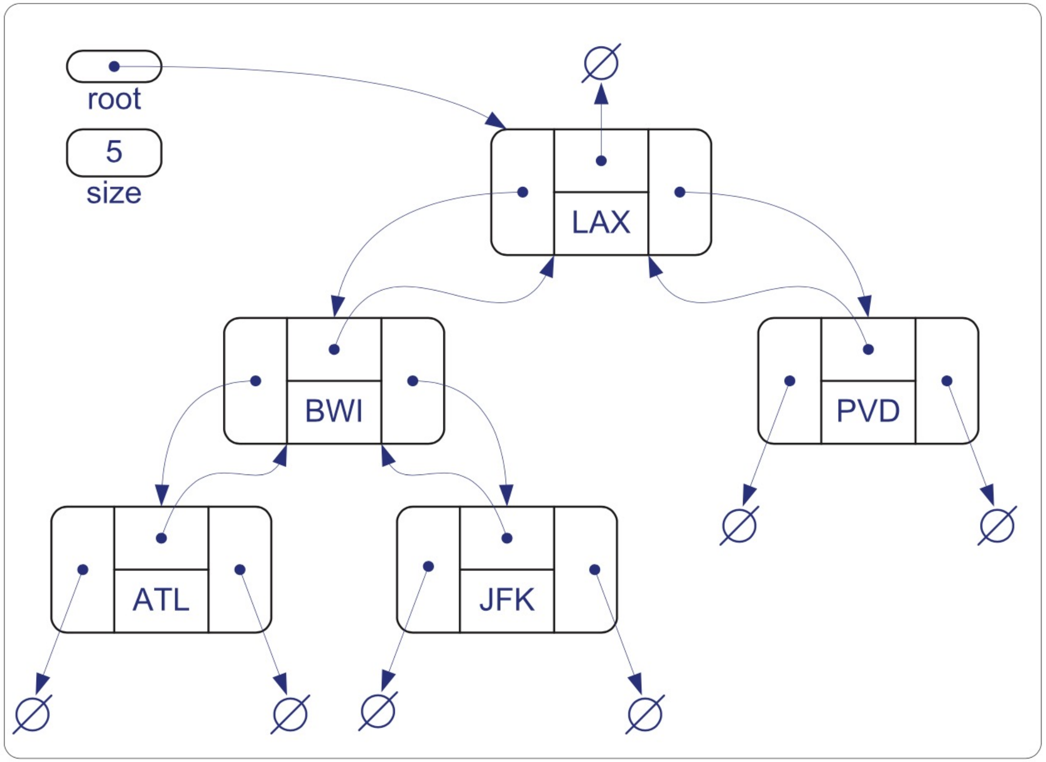
- If T is a proper binary tree:
  - $2h + 1 \leq n \leq 2^{h+1} - 1$
  - $h + 1 \leq n_E \leq 2^h$
  - $h \leq n_I \leq 2^h - 1$
  - $\log(n + 1) - 1 \leq h \leq (n - 1)/2$
  - $n_E = n_I + 1$

BinaryTree ADT

- The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT
- Additional methods:
  - position p.left()
  - position p.right()
- Update methods may be defined by data structures implementing the BinaryTree ADT
  - Proper binary tree: Each node has either 0 or 2 children

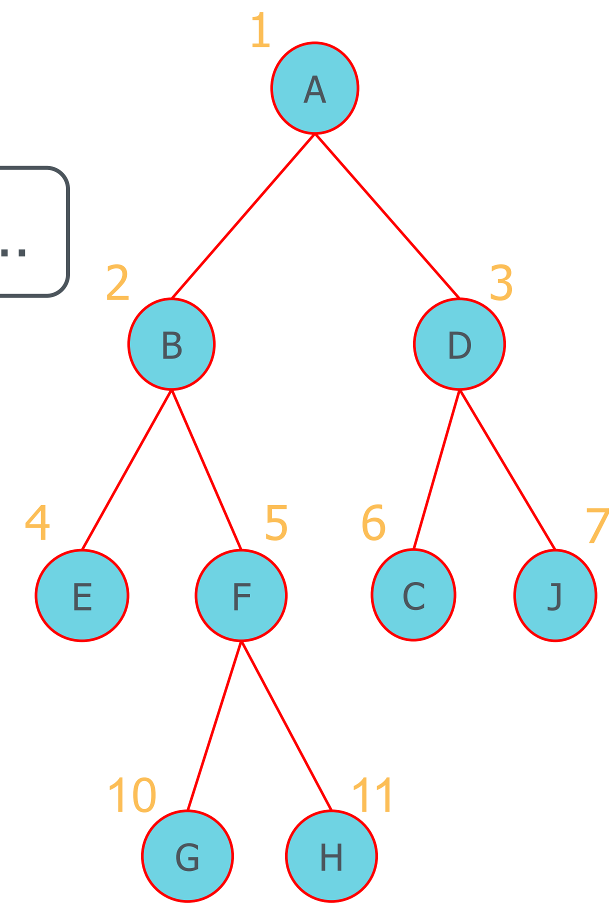
Linked Representation of Binary Trees





Array-Based Representation of Binary Trees

- Nodes are stored in an array A



- ❑ Node v is stored at A[rank(v)]
  - rank(root) = 1
  - if node is the left child of parent(node),
    rank(node) = 2 · rank(parent(node))
  - if node is the right child of parent(node),
    rank(node) = 2 · rank(parent(node)) + 1

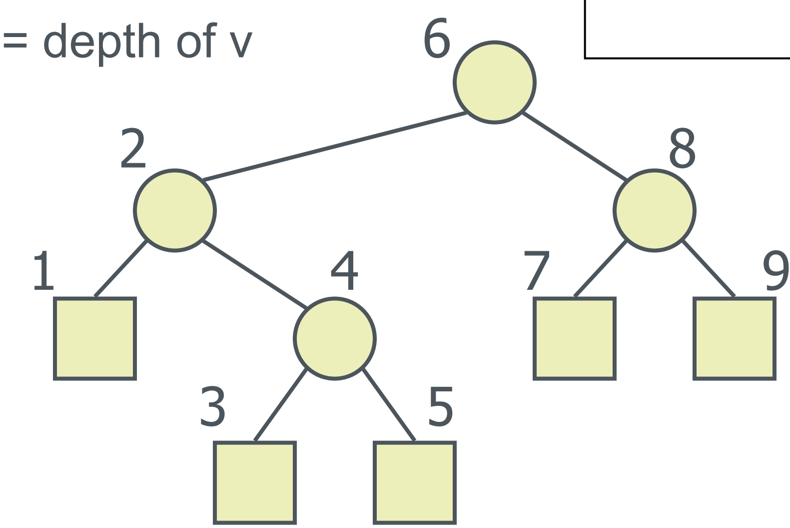
Array-Based Representation of Binary Trees

- Advantages:
  - Suppose n is the number of nodes. The search time can be bounded to $O(\log_2 n)$.
- Disadvantages:
  - The utilization of memory space is not flexible enough.

Inorder Traversal

- In an inorder traversal a node is visited after its left subtree and before its right subtree
- Application: draw a binary tree
  - x(v) = inorder rank of v
  - y(v) = depth of v

Algorithm *inOrder*(*v*)
    if $\neg$ *v.isExternal*()
        inOrder(*v.left*())
    visit(*v*)
    if $\neg$ *v.isExternal*()
        inOrder(*v.right*())



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 *printExpression*(*v*)

    if ¬*v.isExternal*()

        print(“(”)

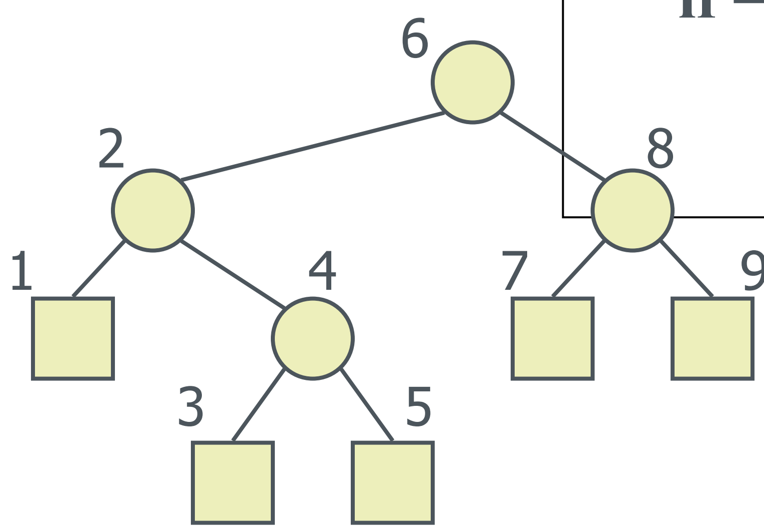
        inOrder(*v.left*())

    print(*v.element*())

    if ¬*v.isExternal*()

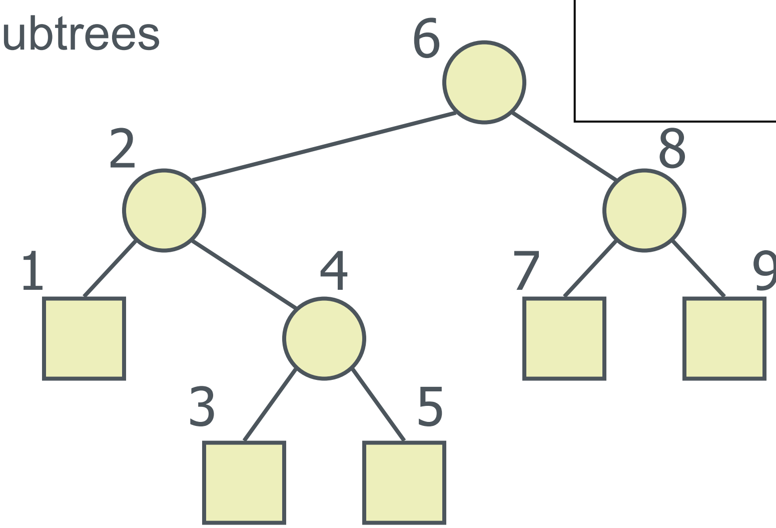
        inOrder(*v.right*())

        print (“)”)



Evaluate Arithmetic Expressions

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



Algorithm ***evalExpr***(v)
    if ***v.isExternal***()
        return ***v.element***()
    else
        $x \leftarrow$ ***evalExpr***(***v.left***())
        $y \leftarrow$ ***evalExpr***(***v.right***())
        $\Diamond \leftarrow$ operator stored at v
        return $x \Diamond y$

Questions?