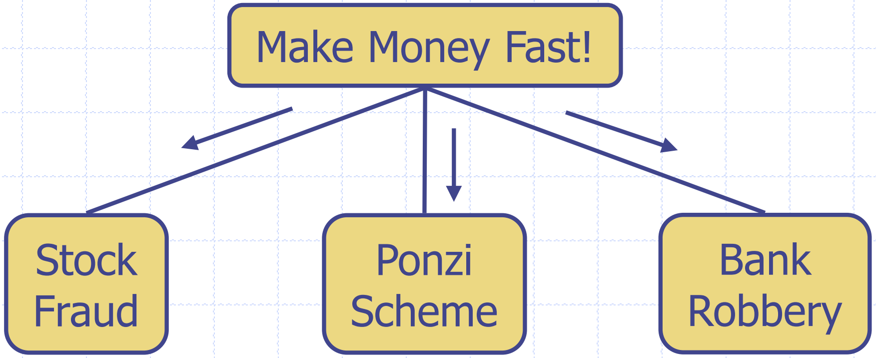
Tree Traversals



Sareh Taebi
COP3410 – Florida Atlantic University

going through entire tree

Tree Traversal

- There are three commonly used patterns to visit all the nodes in a tree.
- The difference between these patterns is the order in which each node is visited.
- The three traversals we will look at are called **preorder**, **inorder**, and **postorder**.

↳ evaluate root first, middle-ish, last

Tree Traversal Algorithms

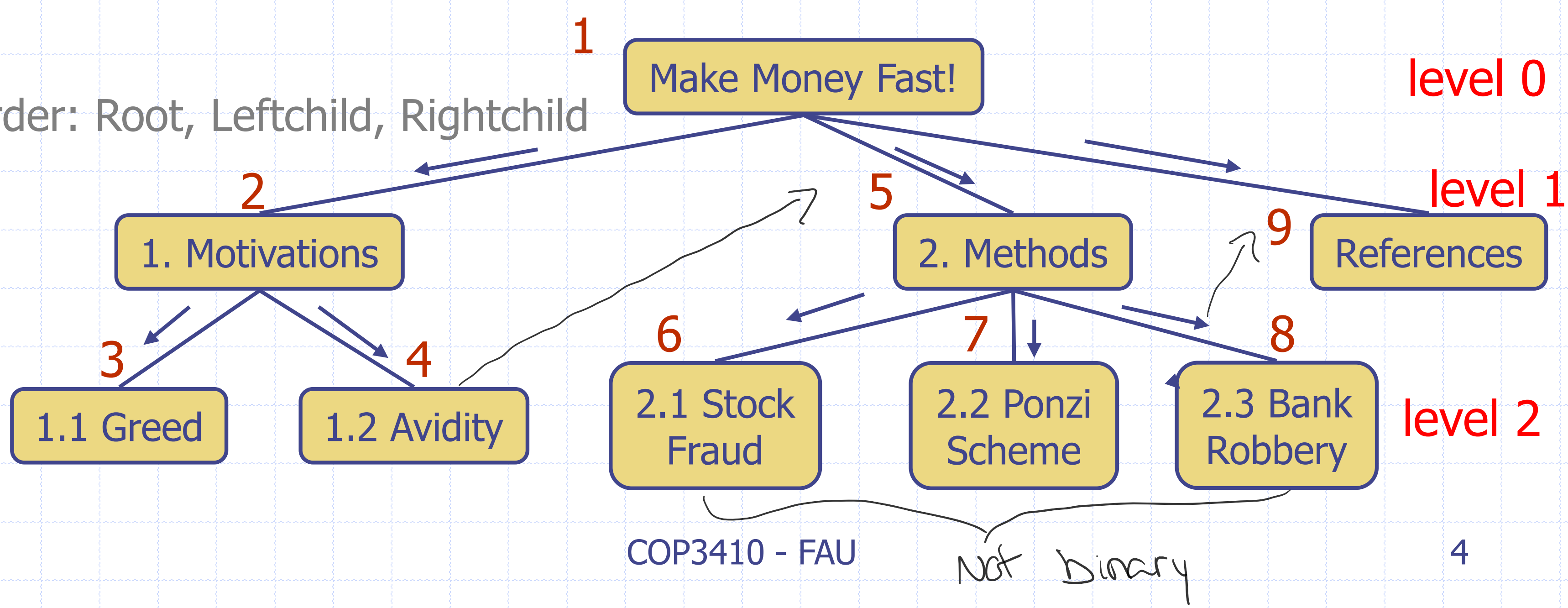
- A ***traversal*** of a tree *T* is a systematic way of accessing, or "visiting" all the nodes of *T*.
- The specific action associated with the "visit" of a position *p* depends on the application of this traversal
- Traverse could involve anything from incrementing a counter to performing some complex computation for that node.

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*)

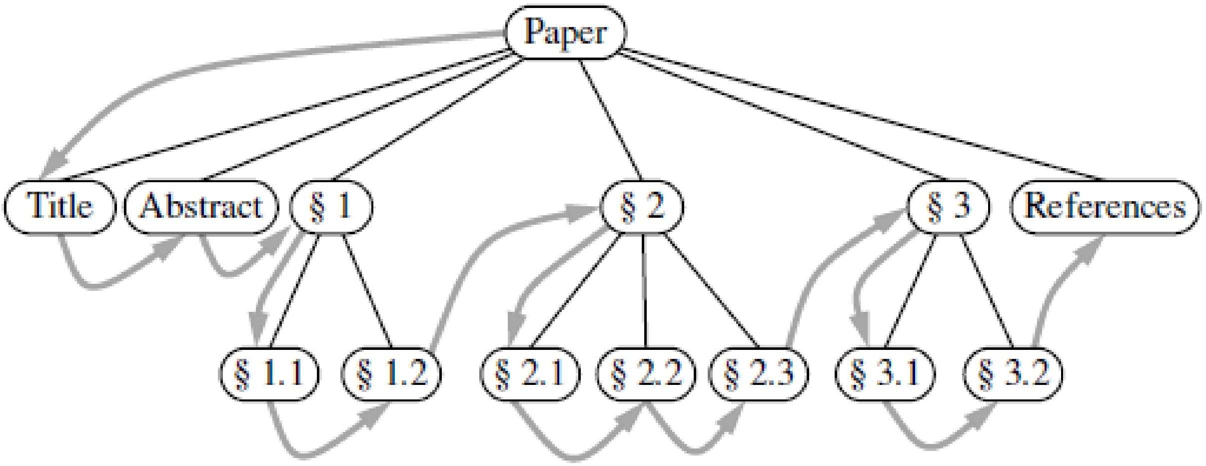
Order: Root, Leftchild, Rightchild



Preorder Traversal

- In a preorder traversal, we visit the root node first, then recursively do a preorder traversal of the descendants from left to right.

print table of contents



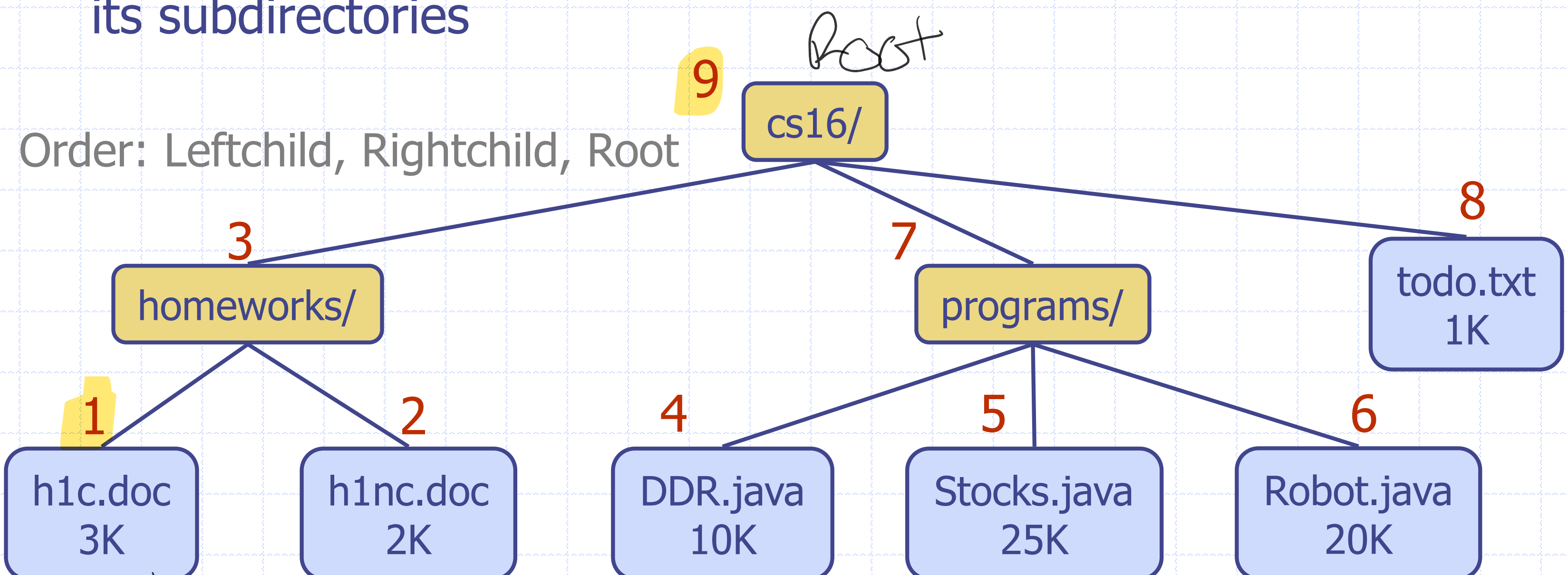
Ordered Tree

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)

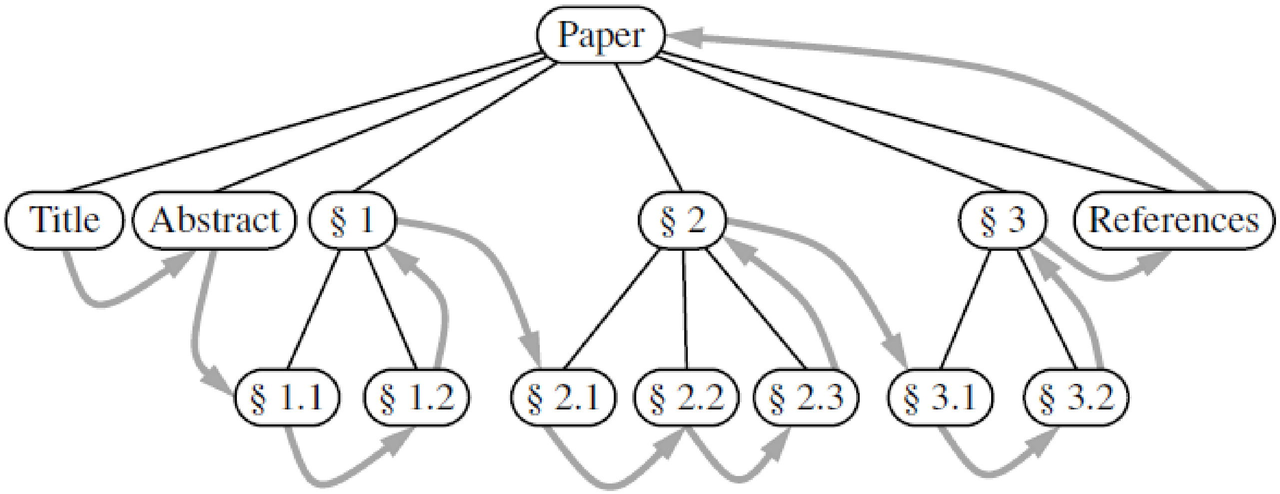
Order: Leftchild, Rightchild, Root



Deepest level first

Postorder Traversal

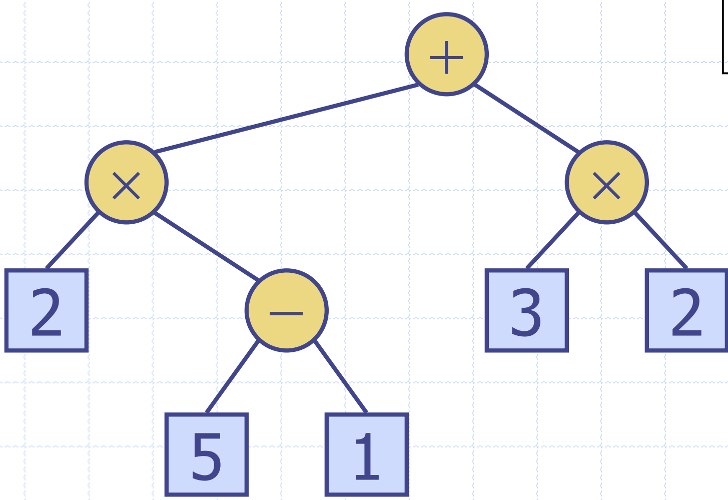
- In a postorder traversal, we recursively do a postorder traversal of the descendants from left to right followed by a visit to the root node.



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

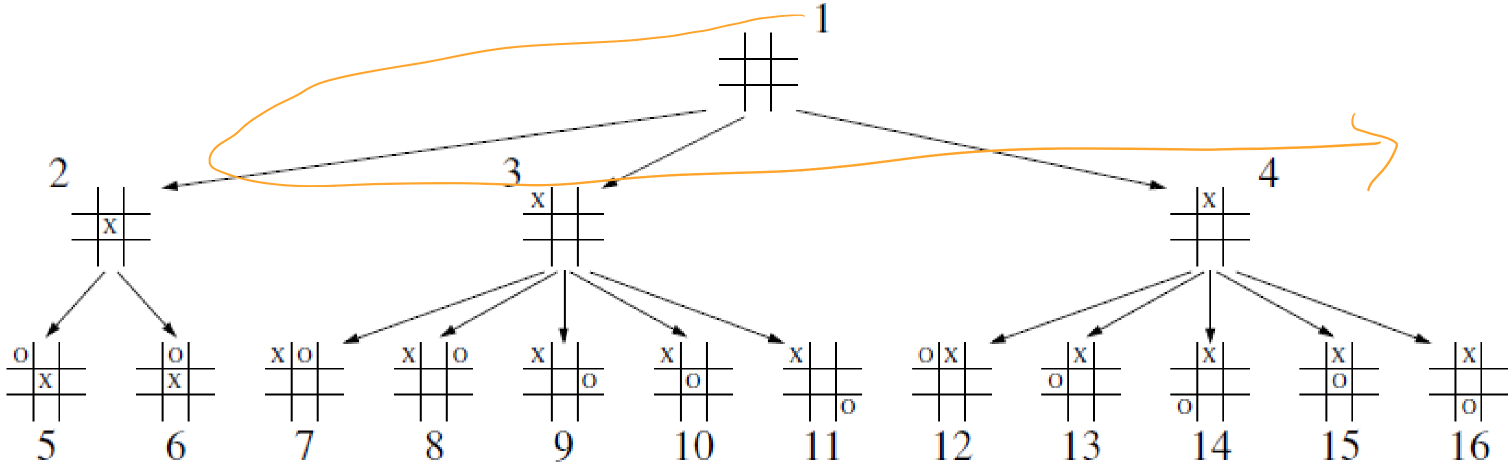
Algorithm *evalExpr*(v)
    if *is_leaf* (v)
        return *v.element* ()
    else
        $x \leftarrow$ *evalExpr*(*left* (v))
        $y \leftarrow$ *evalExpr*(*right* (v))
        $\Diamond \leftarrow$ operator stored at v
        return $x \Diamond y$



$2 \times (5-1) + (3 \times 2)$

Breadth First Traversal (BFT)

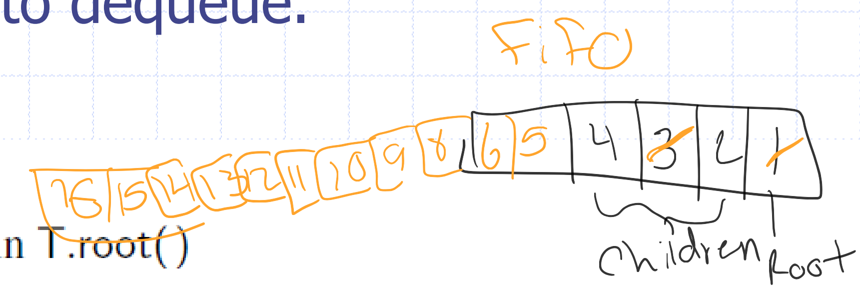
- We visit all the positions at depth `d` before we visit the positions at depth `d+1`.
- Such an algorithm is known as a breadth-first traversal.



Game tree: All possible moves by the player/computer

Breadth First Traversal (BFT)

- The BFT algorithm is *not* recursive, since we are not traversing entire subtrees at once.
- We use a queue to produce a FIFO semantics for the order of nodes visited.
- The overall running time is $O(n)$, due to the n calls to enqueue and n calls to dequeue.



Algorithm breadthfirst(T):
  Initialize queue Q to contain T.root()
  while Q not empty **do**
    p = Q.dequeue()   {p is the oldest entry in the queue}
    perform the "visit" action for position p
    for each child c in T.children(p) **do**
      Q.enqueue(c)   {add p's children to the end of the queue for later visits}

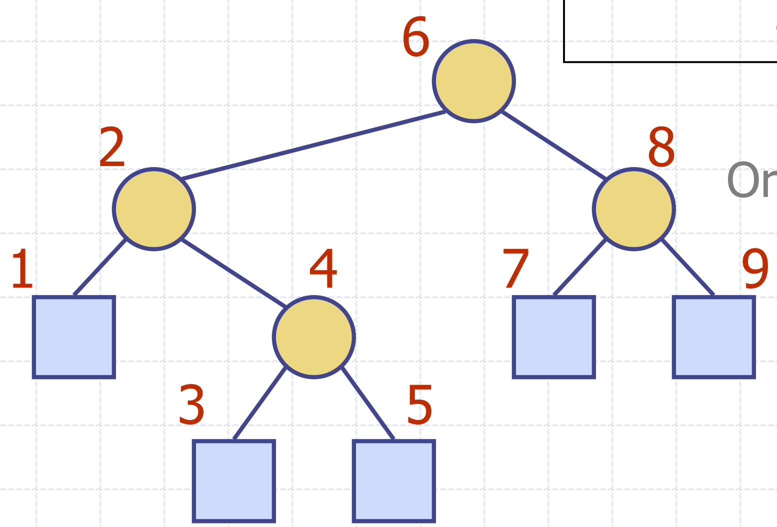
Traversing every level
↳searching for each level

Inorder Traversal

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

↳ use for binary trees
↳ others are universal

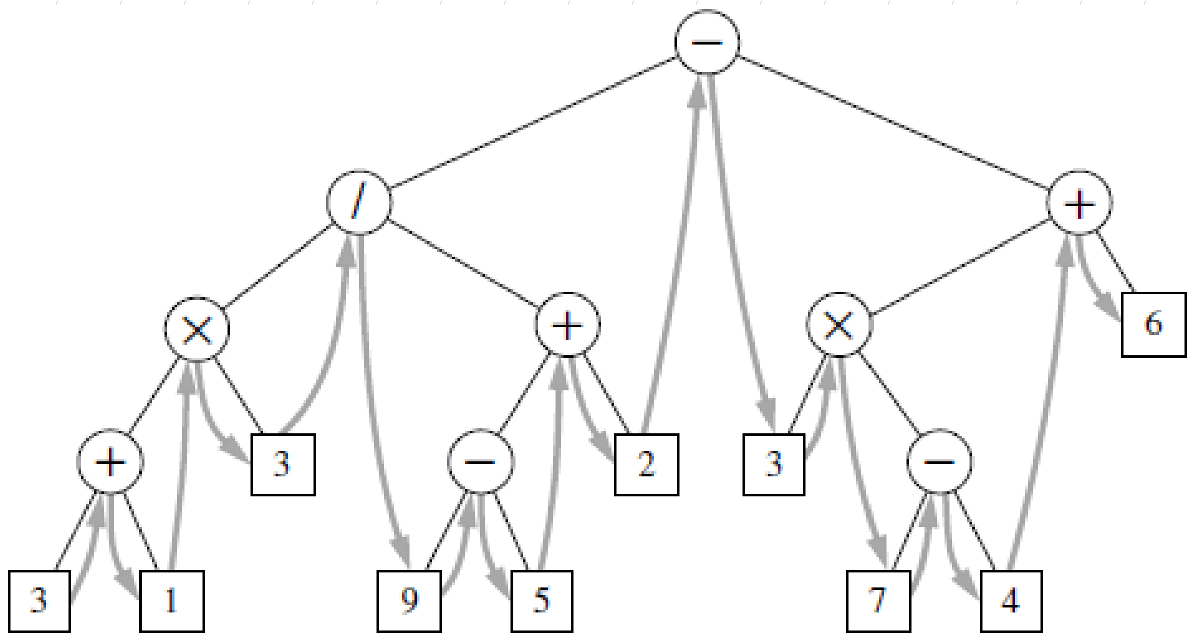
Algorithm *inOrder*(*v*)
    if *v* **has a left child**
        inOrder (*left* (*v*))
    visit(*v*)
    if *v* **has a right child**
        inOrder (*right* (*v*))



Order: Leftchild, Root, Rightchild

↳ Recursion

Inorder Traversal of a Binary Tree



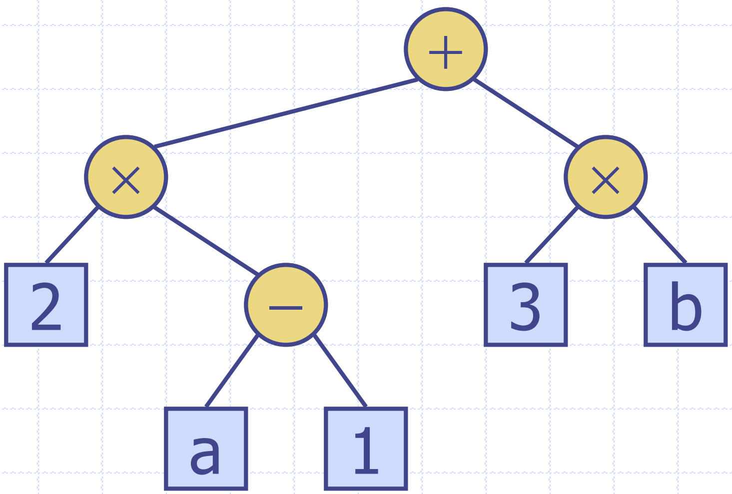
Important natural application: arithmetic expressions

3+1 × 3 / 9−5+2 . . .

No parenthesis

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* **has a left child**
    print(“(’ ’)
    inOrder (*left*(*v*))
  print(*v.element* ())
  if *v* **has a right child**
    inOrder (*right*(*v*))
    print (“)’ ’)

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

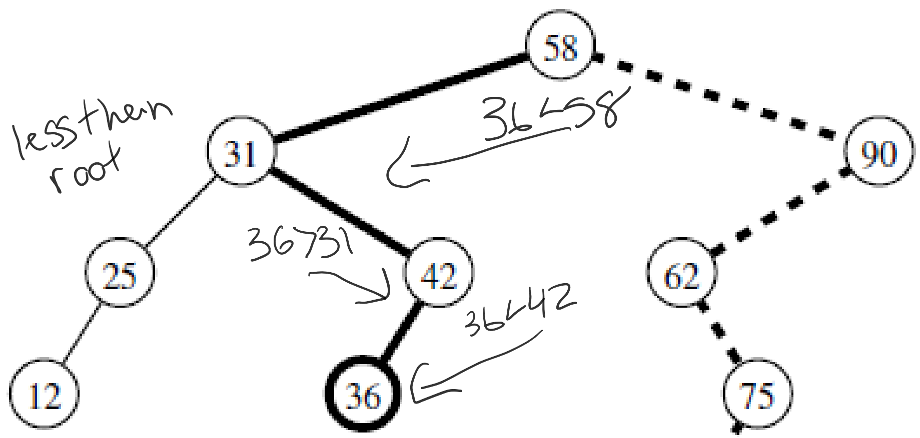
Exam & Hw

Binary Search Tree

- Important application of the inorder traversal
- A binary search tree for S is a binary tree T such that, for each position p of T:
  - Position p stores an element of S, denoted as $e(p)$.
  - Elements stored in the left subtree of p (if any) are less than $e(p)$.
  - Elements stored in the right subtree of p (if any) are greater than $e(p)$.

Populate left first

less then root



greater than root

search for 36 (successful)

search for 70 (unsuccessful)

running time proportional to the height of tree with n nodes:

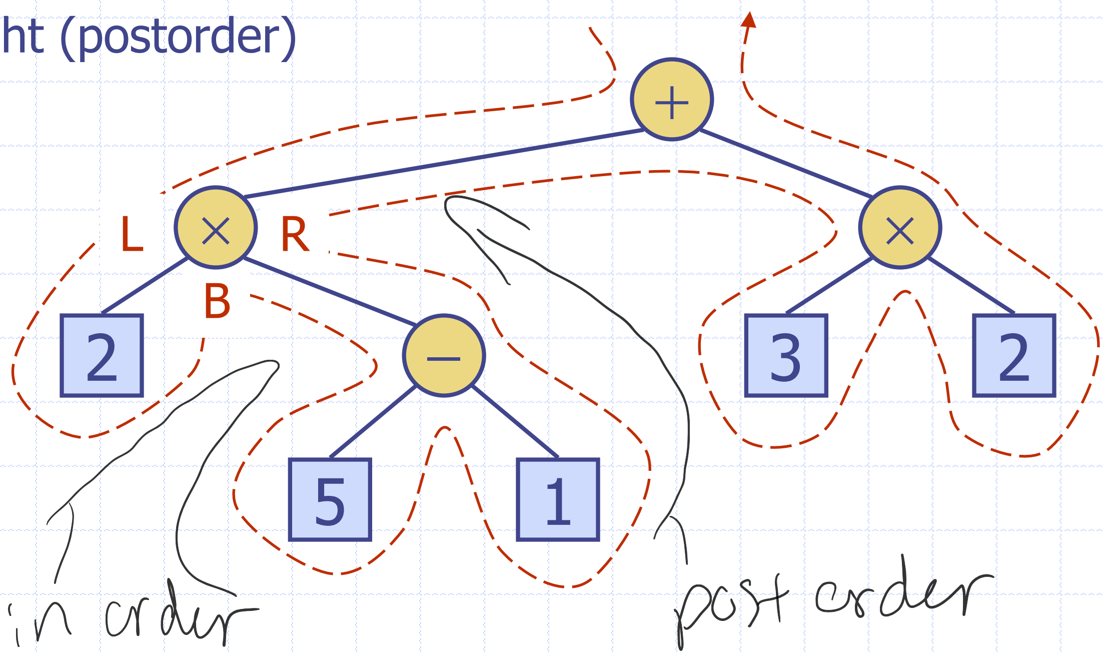
$\log(n+1)-1 \le h \le n-1$ why? spread

Euler Tour Traversal

- Generic traversal of a binary tree
- Includes a special cases the preorder, postorder and inorder traversals
- Walk around the tree and visit each node three times:
  - on the left (preorder)
  - from below (inorder)
  - on the right (postorder)

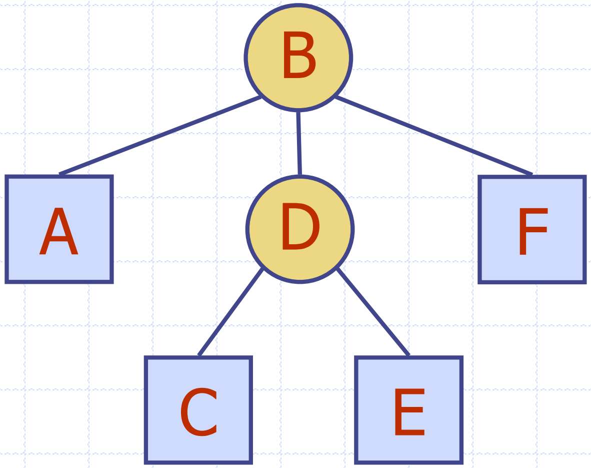
O(1)

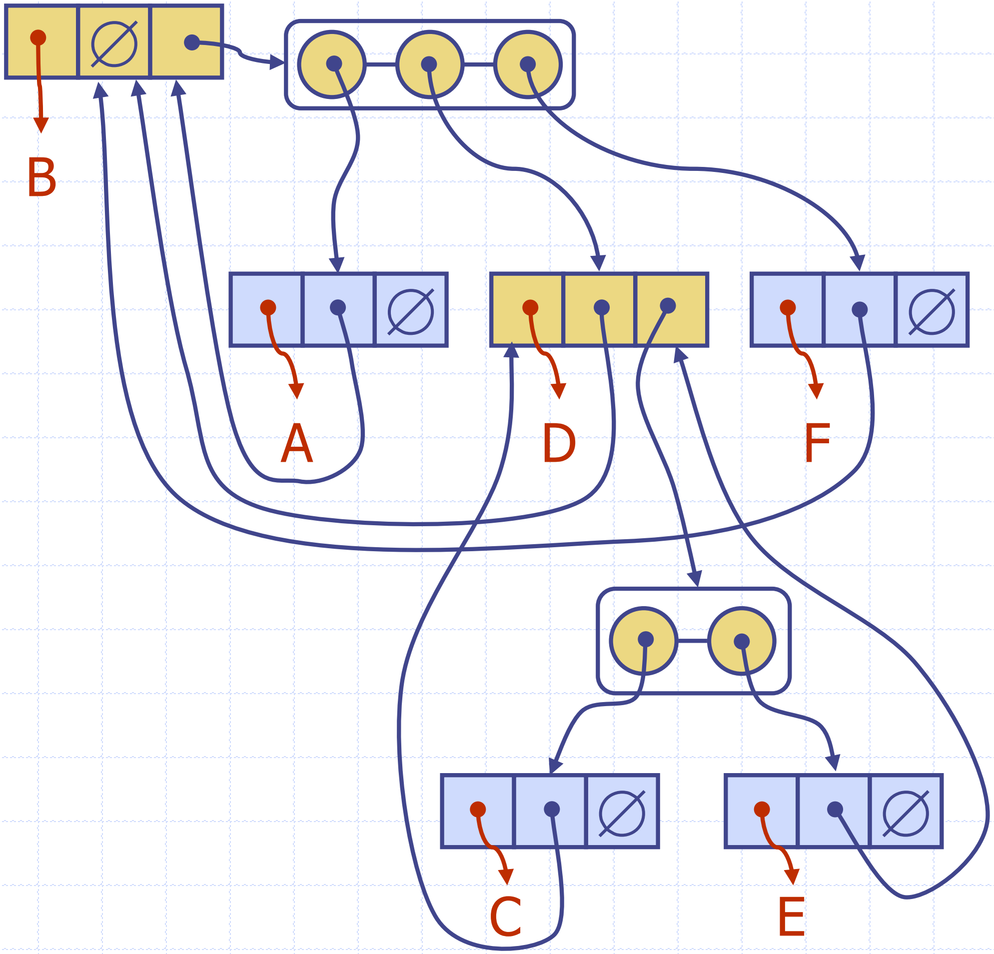
visits twice
↳ O(2n)
↳ O(n)



Linked Structure for Trees

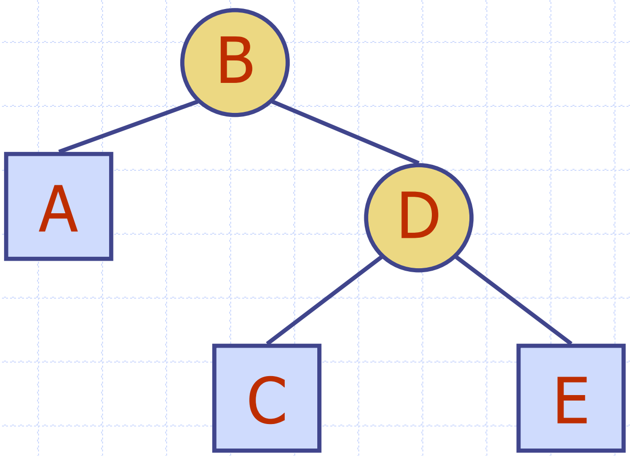
- A node is represented by an object storing
  - Element
  - Parent node
  - Sequence of children nodes
- Node objects implement the Position ADT

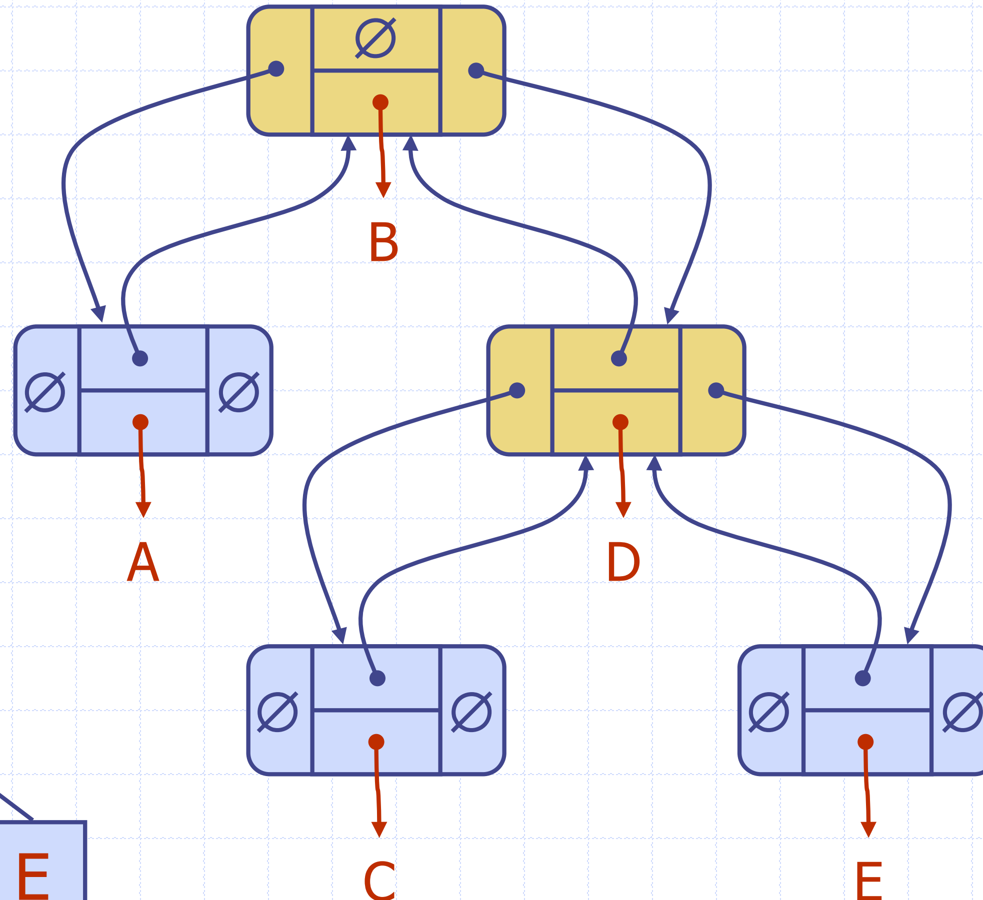




Linked Structure for Binary Trees

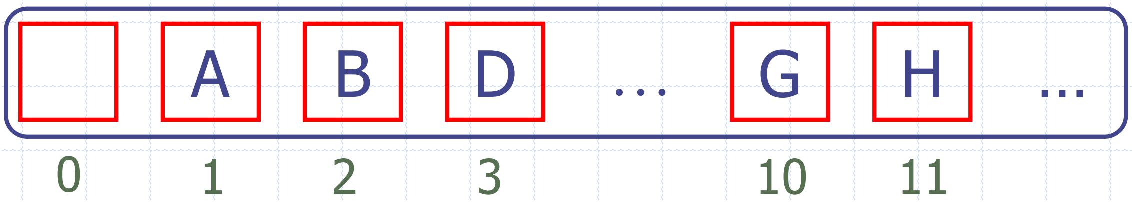
- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node
- Node objects implement the Position ADT





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