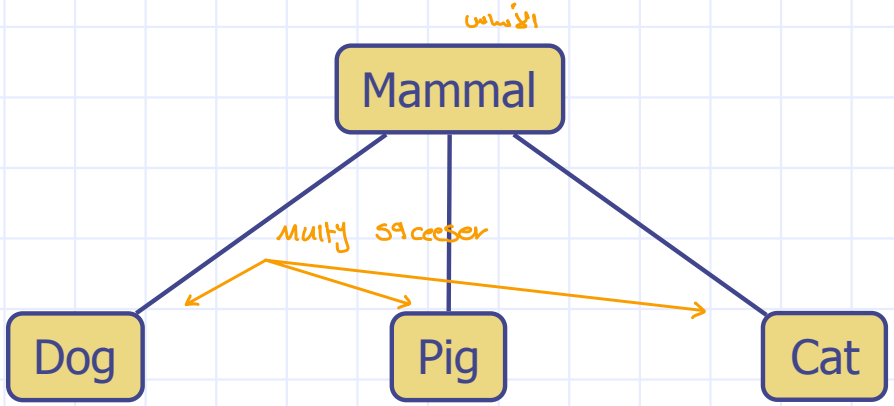
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

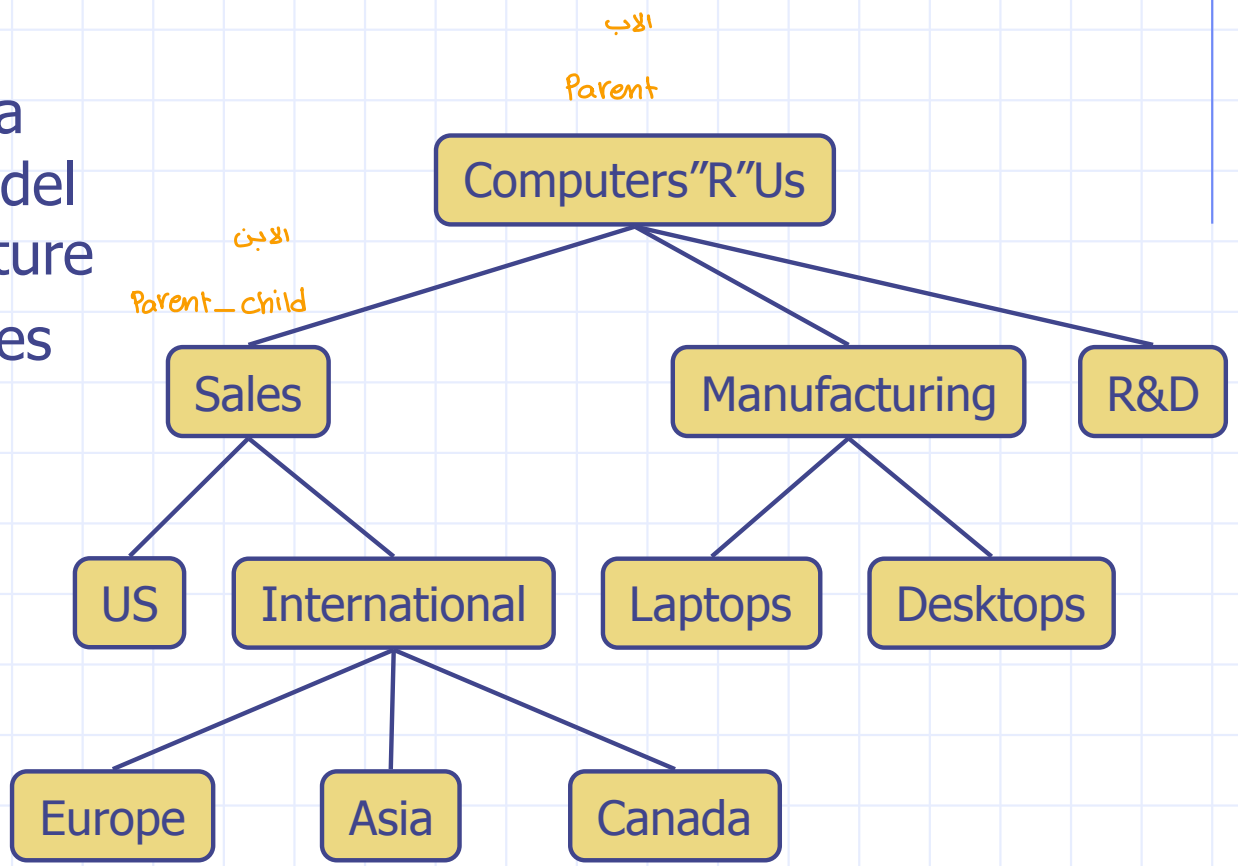
Trees

يبدأ من فوق لتحت.



What is a Tree

- In computer science, a tree is an abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation
- Applications:
  - Organization charts
  - File systems
  - Programming environments



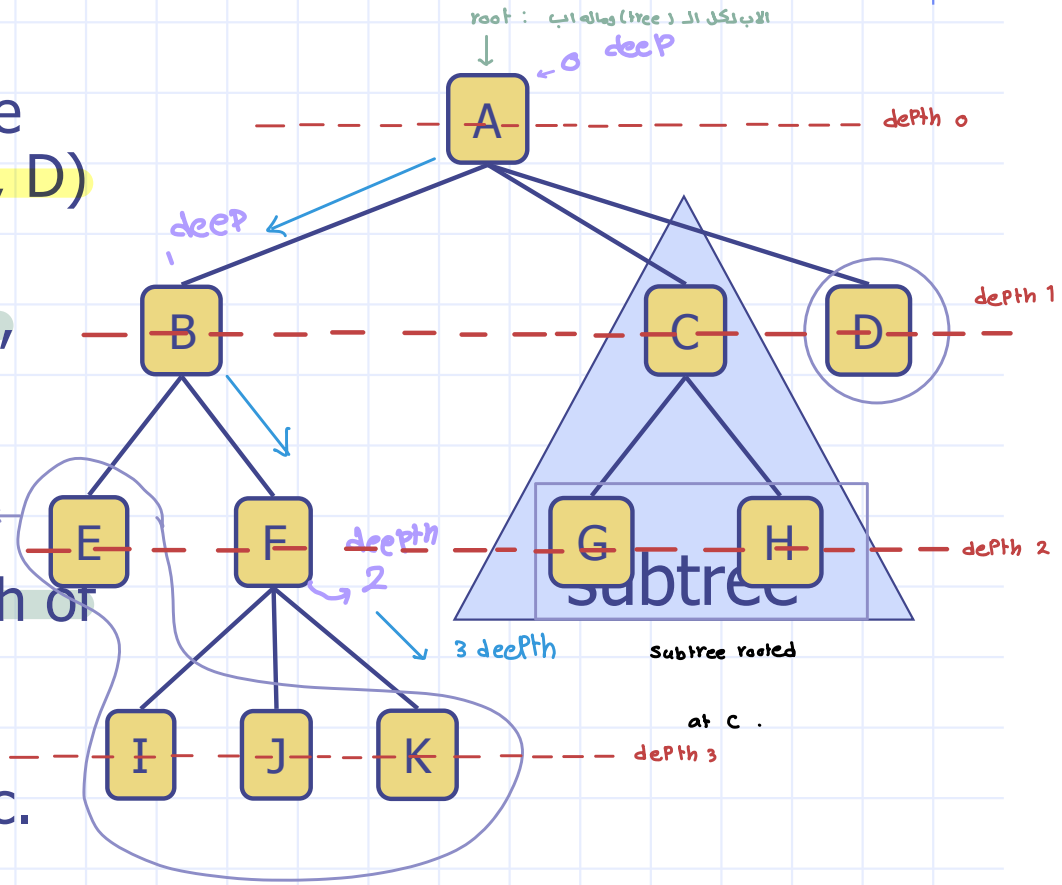
Tree Terminology

Parent
Child

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

★ Siblings : أخوان من نفس الأب

- Subtree: tree consisting of a node and its descendants



Tree ADT

- We use positions to abstract nodes
- Generic methods:
  - integer size()
  - boolean isEmpty()
  - Iterator iterator()
  - Iterable positions()
- Accessor methods:
  - position root()
  - position parent(p)
  - Iterable children(p)
  - Integer numChildren(p)

- Query methods:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)

- Additional update methods may be defined by data structures implementing the Tree ADT

Java Interface

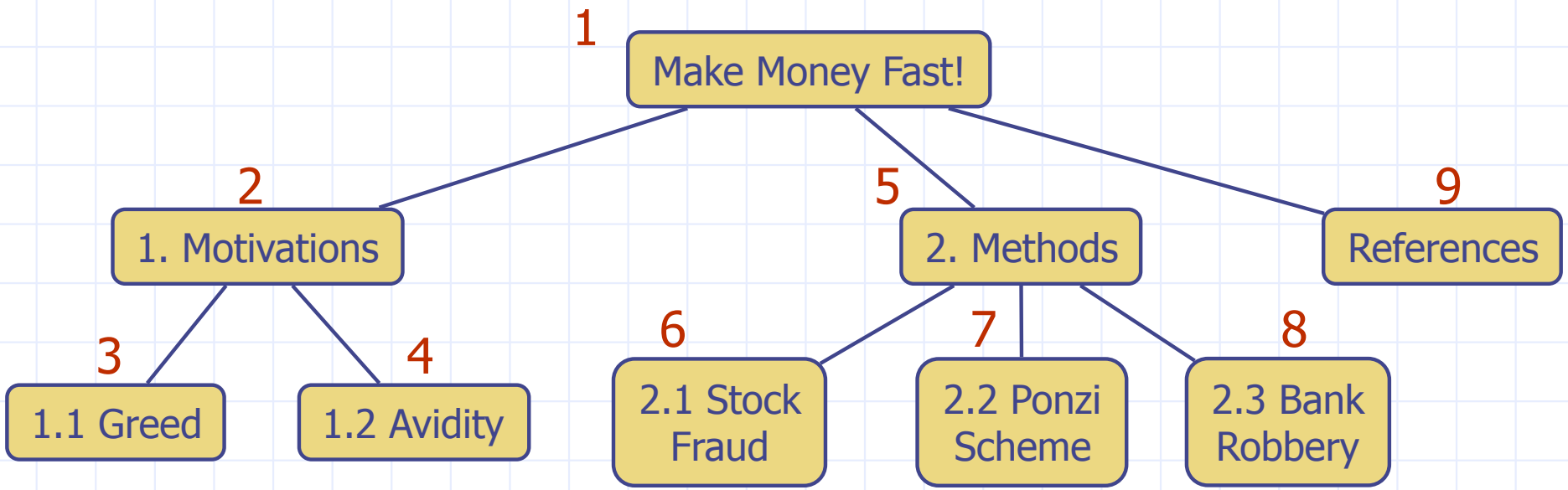
Methods for a Tree interface:

```
1  /** An interface for a tree where nodes can have an arbitrary number of children. */
2  public interface Tree<E> extends Iterable<E> {
3    Position<E> root( );
4    Position<E> parent(Position<E> p) throws IllegalArgumentException;
5    Iterable<Position<E>> children(Position<E> p)
6                                          throws IllegalArgumentException;
7    int numChildren(Position<E> p) throws IllegalArgumentException;
8    boolean isInternal(Position<E> p) throws IllegalArgumentException;
9    boolean isExternal(Position<E> p) throws IllegalArgumentException;
10   boolean isRoot(Position<E> p) throws IllegalArgumentException;
11   int size( );
12   boolean isEmpty( );
13   Iterator<E> iterator( );
14   Iterable<Position<E>> positions( );
15 }
```

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

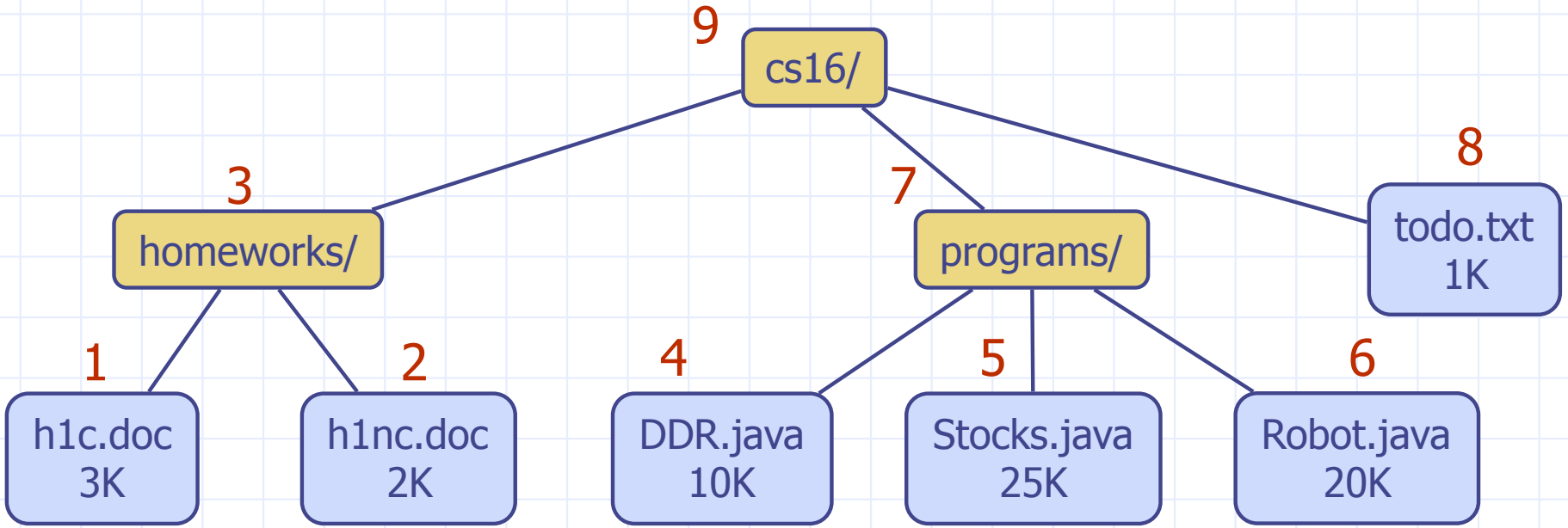
Algorithm *preOrder(v)*
    visit(v)
    for each child *w* of *v*
        preorder (w)



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

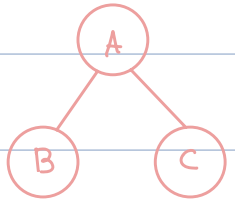
Algorithm *postOrder(v)*
  for each child *w* of *v*
    postOrder (w)
  visit(v)



inorder : a + b

Preorder : + a b

Postorder : a b +



In : B A C

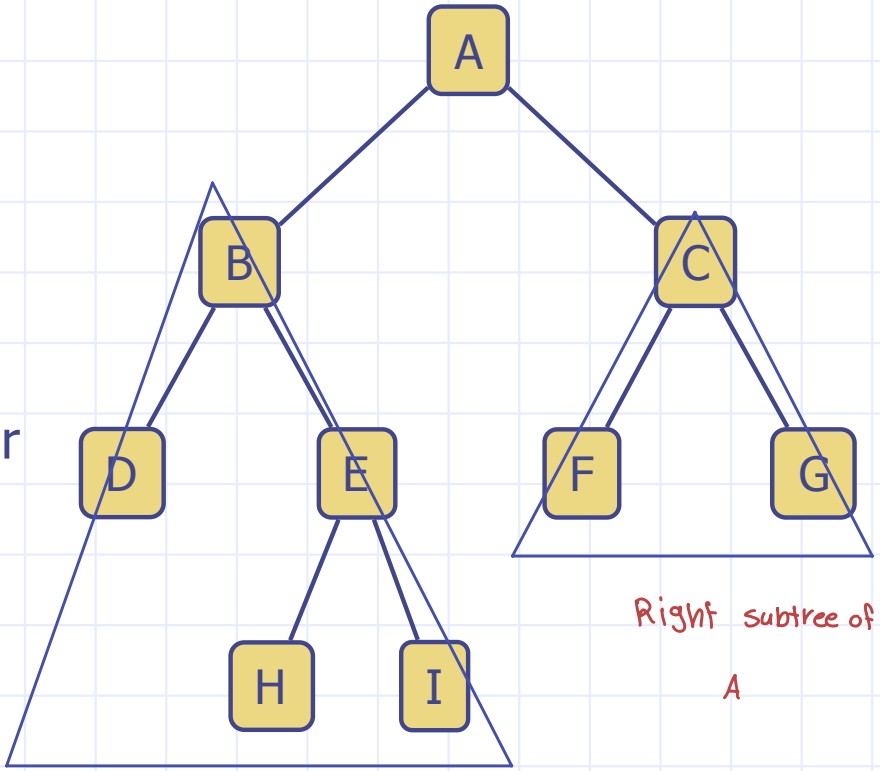
Pre : A B C

Post : B C A

Binary Trees

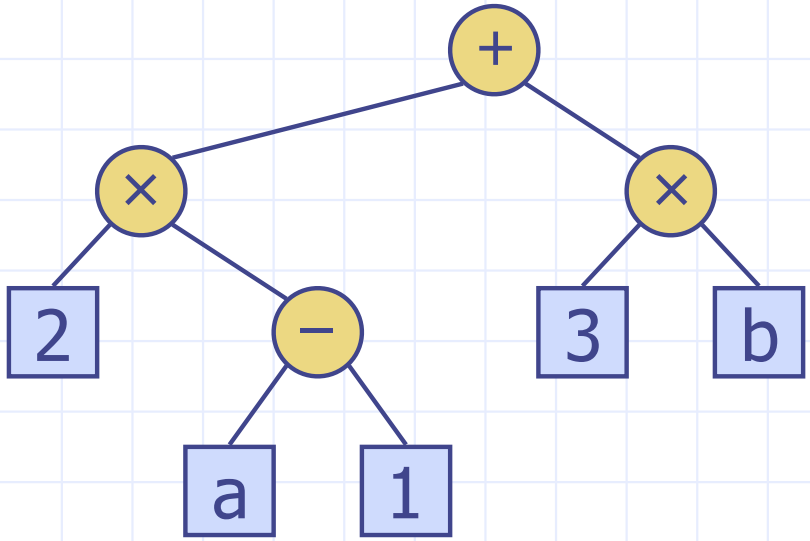
- A binary tree is a tree with the following properties:
  - Each internal node has at most two children (exactly two for proper binary trees)
  - The children of a node are an ordered pair
- We call the children of an internal node left child and right child
- Alternative recursive definition: a binary 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



Arithmetic Expression Tree

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



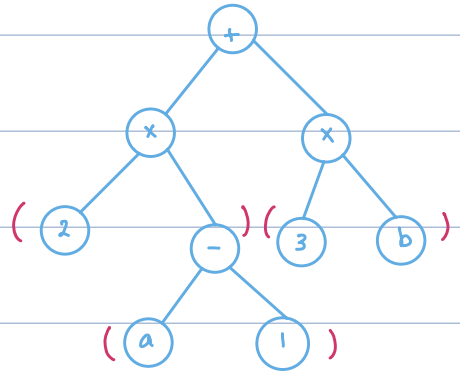
((2x(a-1)+(3xb))

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

كل ما لاقيت قوس مفتوح بنزل level.

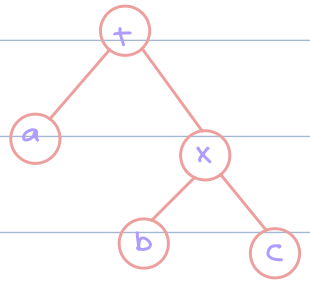
كل ما لاقيت قوس مقفل بطلع level.

معصة

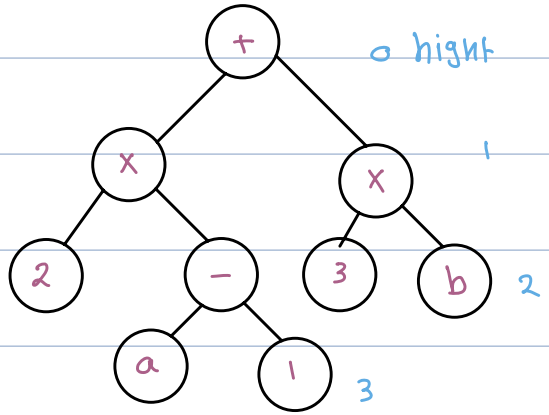


((2x(a-1)) + (3xb))

(a + (bxc))



(a+ (bxc))



0 hight

1

2

3

Inorder : L and R

2 x a - 1 + 3 x b

Preorder : root L R

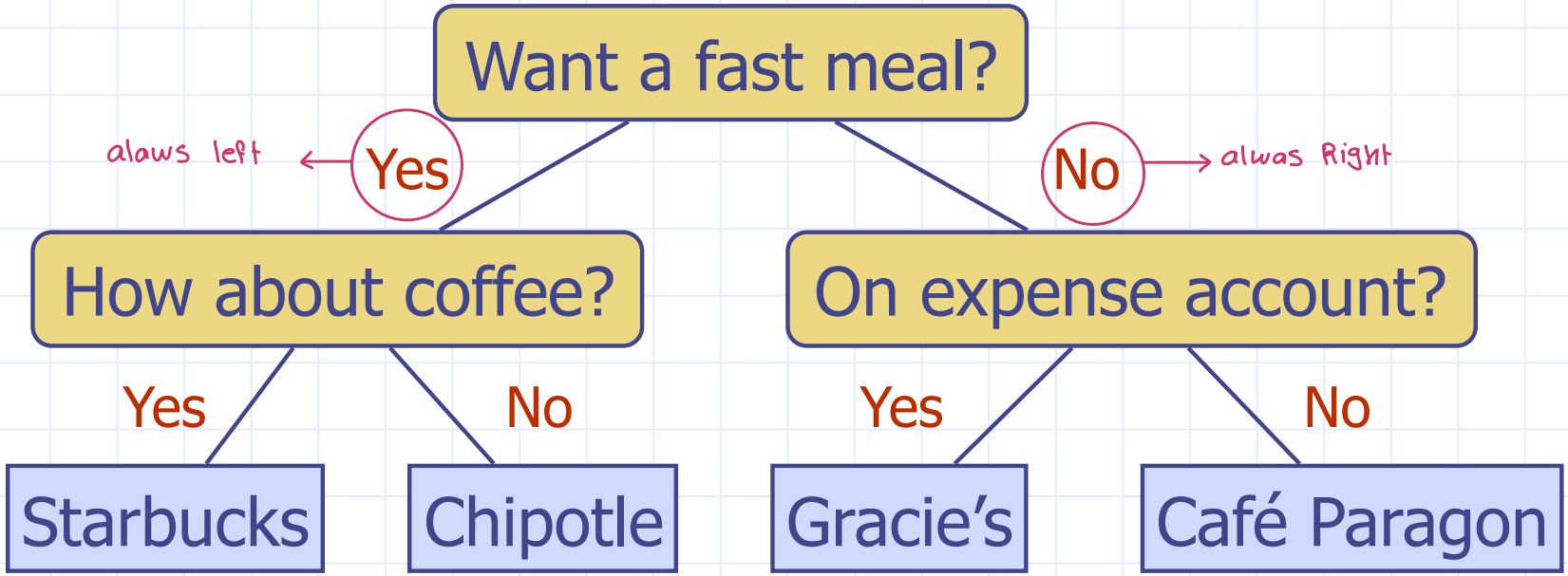
+ x 2 - a 1 x 3 b

Postorder : L R root

2 a 1 - x 3 b x +

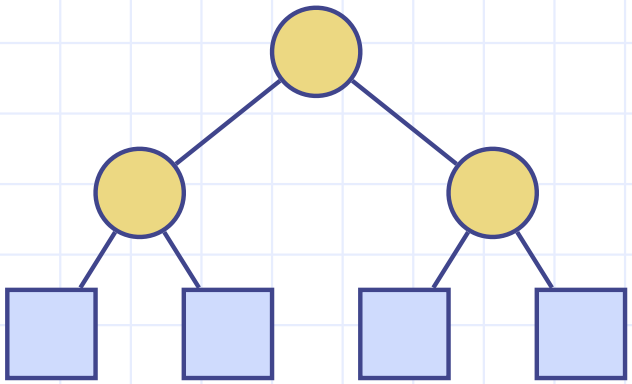
Decision Tree

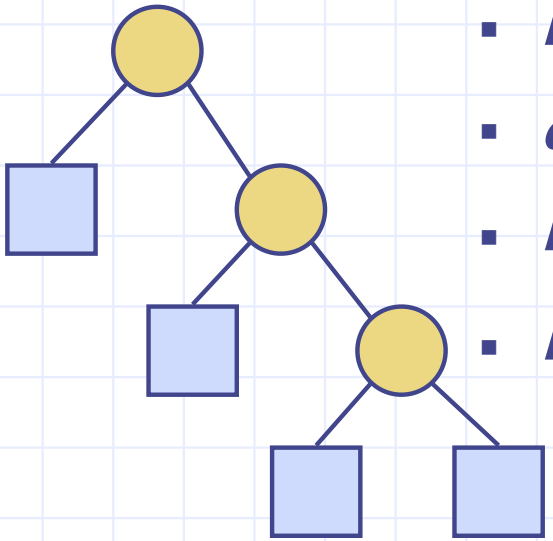
- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision



Properties of Proper Binary Trees

- Notation
  - n number of nodes
  - e number of external nodes
  - i number of internal nodes
  - h height





- Properties:
  - $e = i + 1$
  - $n = 2e - 1$
  - $h \leq i$
  - $h \leq (n - 1)/2$
  - $e \leq 2^h$
  - $h \geq \log_2 e$
  - $h \geq \log_2 (n + 1) - 1$

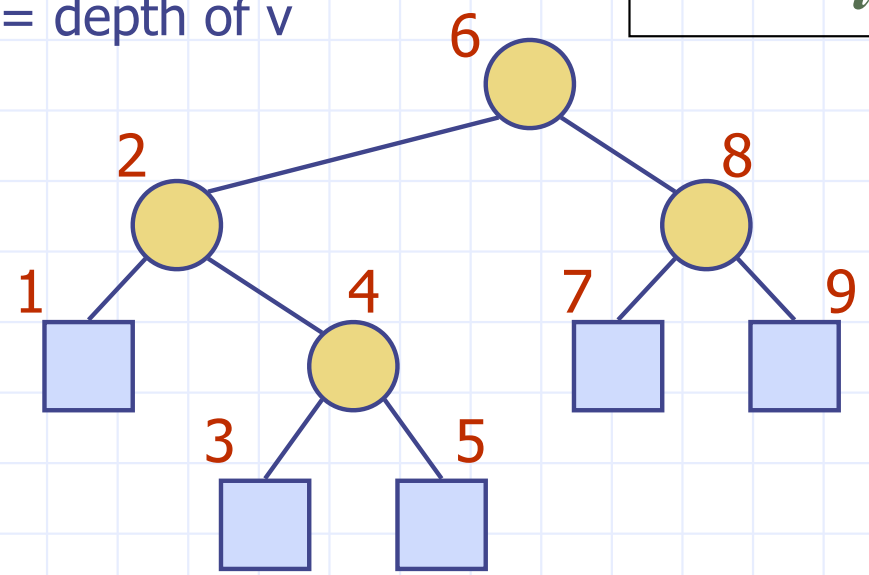
BinaryTree ADT

- The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT
- Additional methods:
  - position left(p)
  - position right(p)
  - position sibling(p)
- The above methods return null when there is no left, right, or sibling of p, respectively
- Update methods may be defined by data structures implementing the BinaryTree ADT

Inorder Traversal

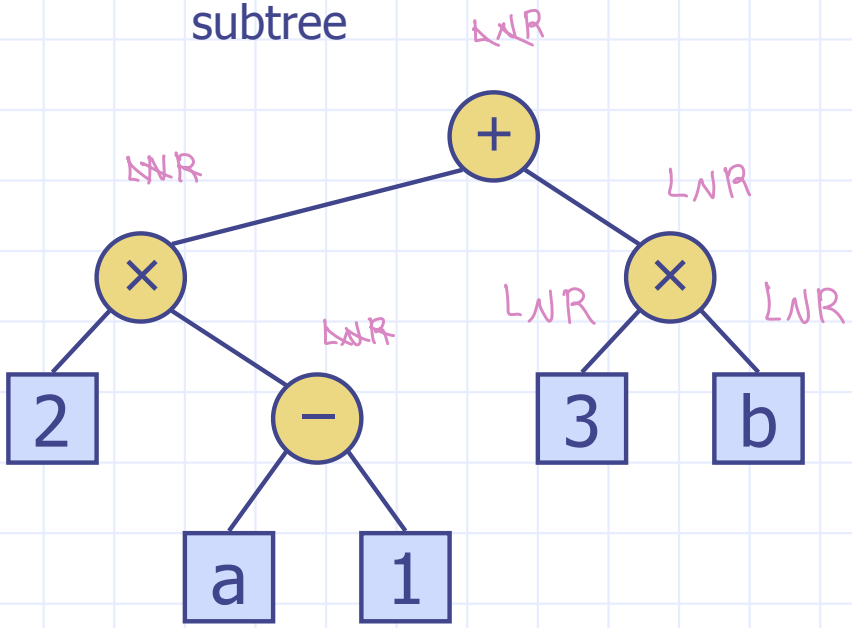
- In an inorder traversal a node is visited after its left subtree and before its right subtree
- Application: draw a binary tree
  - x(v) = inorder rank of v
  - y(v) = depth of v

Algorithm *inOrder*(*v*)
  if *left* (*v*) ≠ **null**
    inOrder (*left* (*v*))
  visit(*v*)
  if *right*(*v*) ≠ **null**
    inOrder (*right* (*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

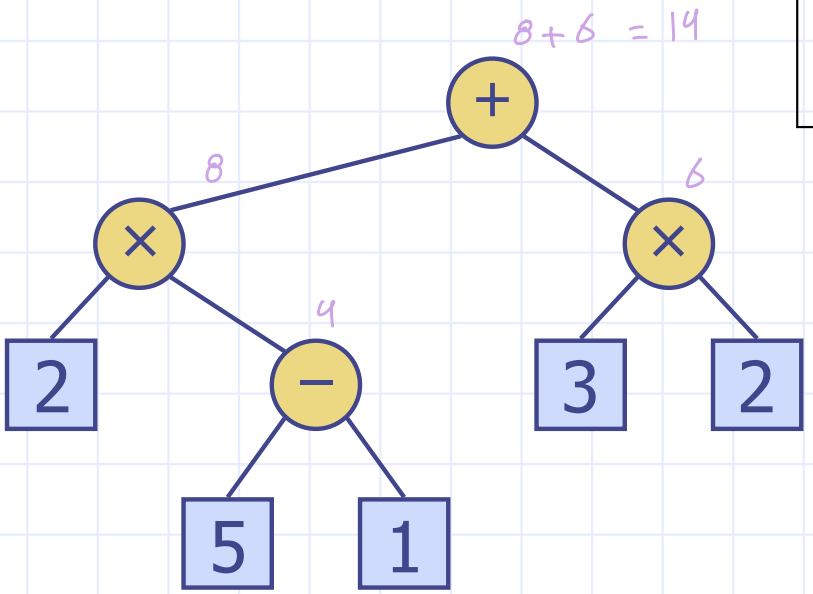


Algorithm *printExpression*(*v*)
  if *left* (*v*) ≠ **null**
    print("(")
    inOrder (*left*(*v*))
  print(*v.element* ())
  if *right*(*v*) ≠ **null**
    inOrder (*right*(*v*))
    print (")")

((2 × (a – 1)) + (3 × b))

Evaluate Arithmetic Expressions

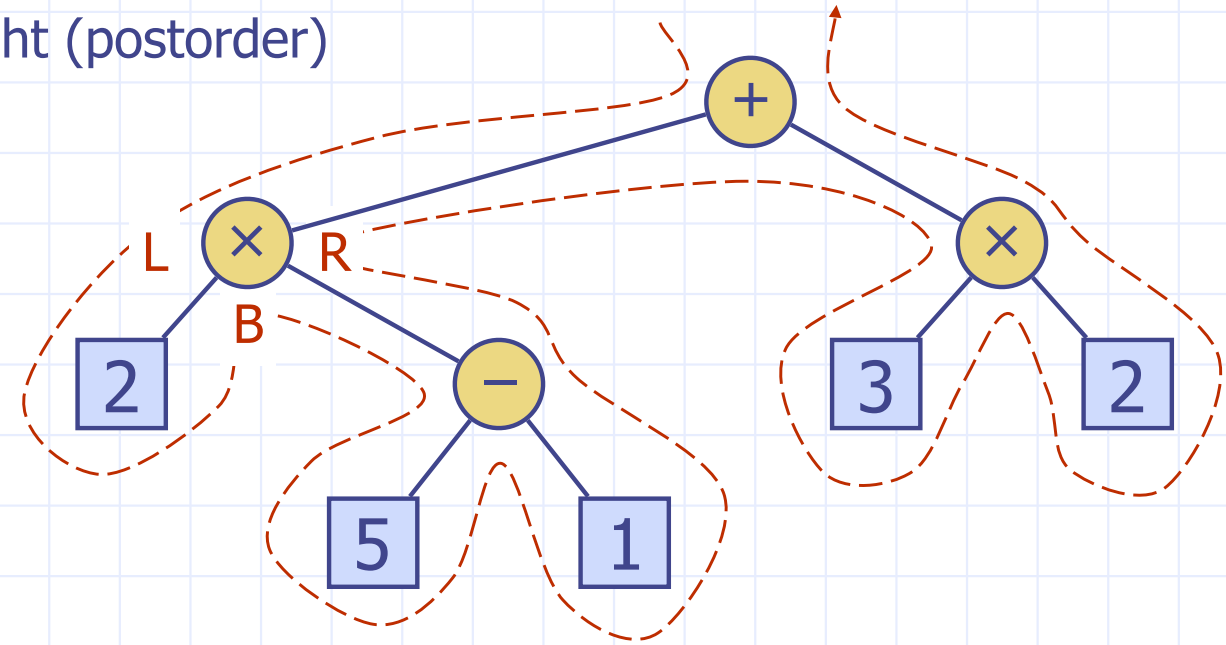
- Specialization of a postorder traversal
  - recursive method returning the value of a subtree
  - when visiting an internal node, combine the values of the subtrees



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

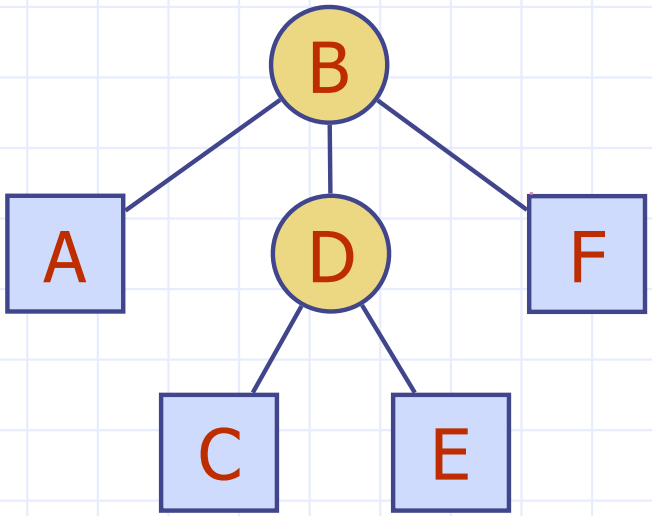
Euler Tour Traversal

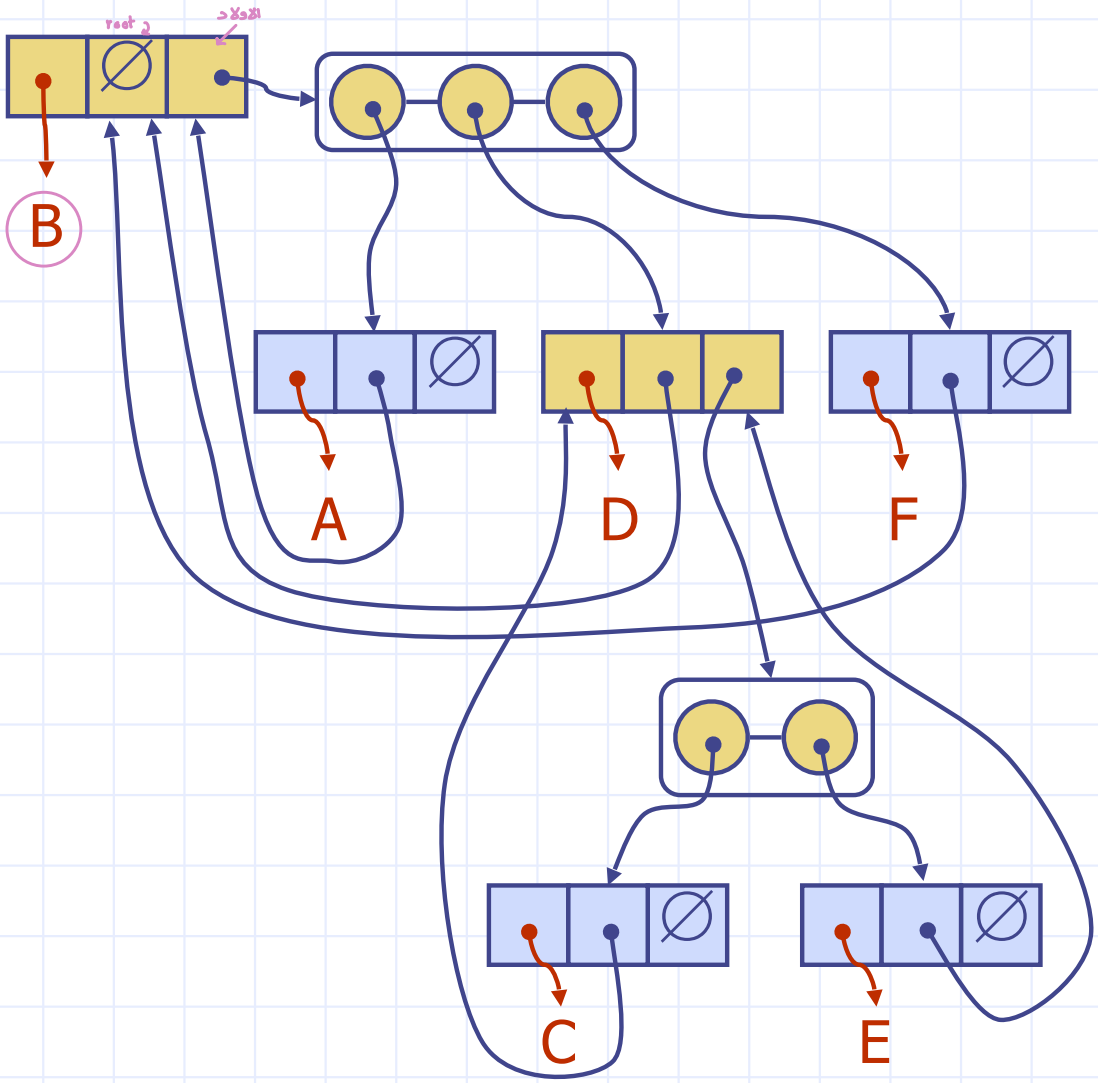
- Generic traversal of a binary tree
- Includes a special cases the preorder, postorder and inorder traversals
- Walk around the tree and visit each node three times:
  - on the left (preorder)
  - from below (inorder)
  - on the right (postorder)

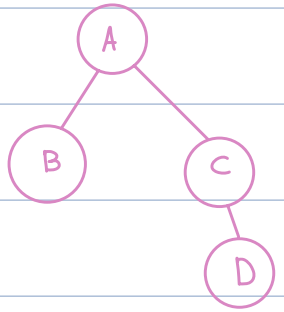


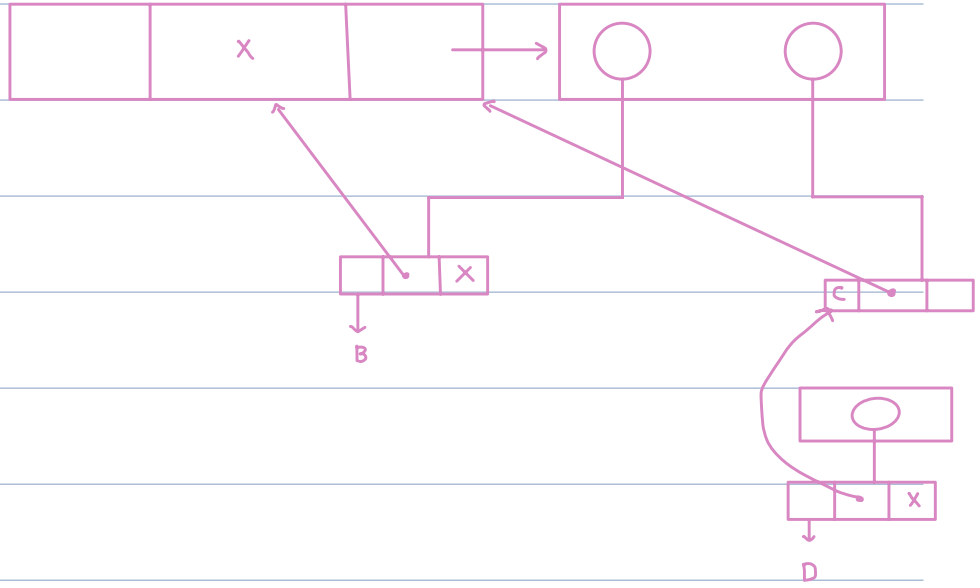
Linked Structure for Trees

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



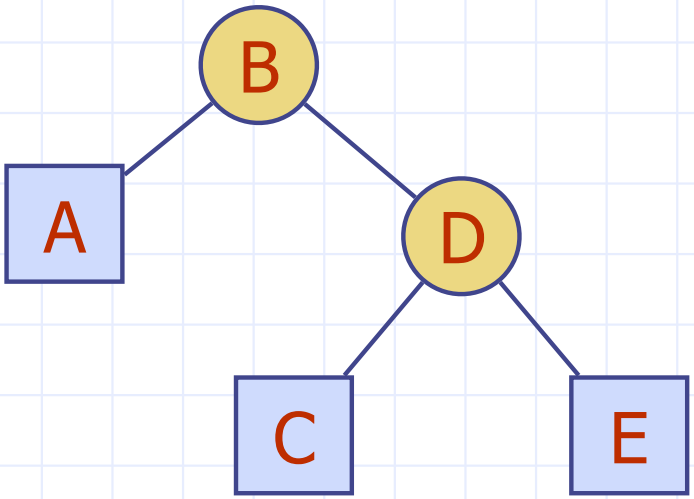


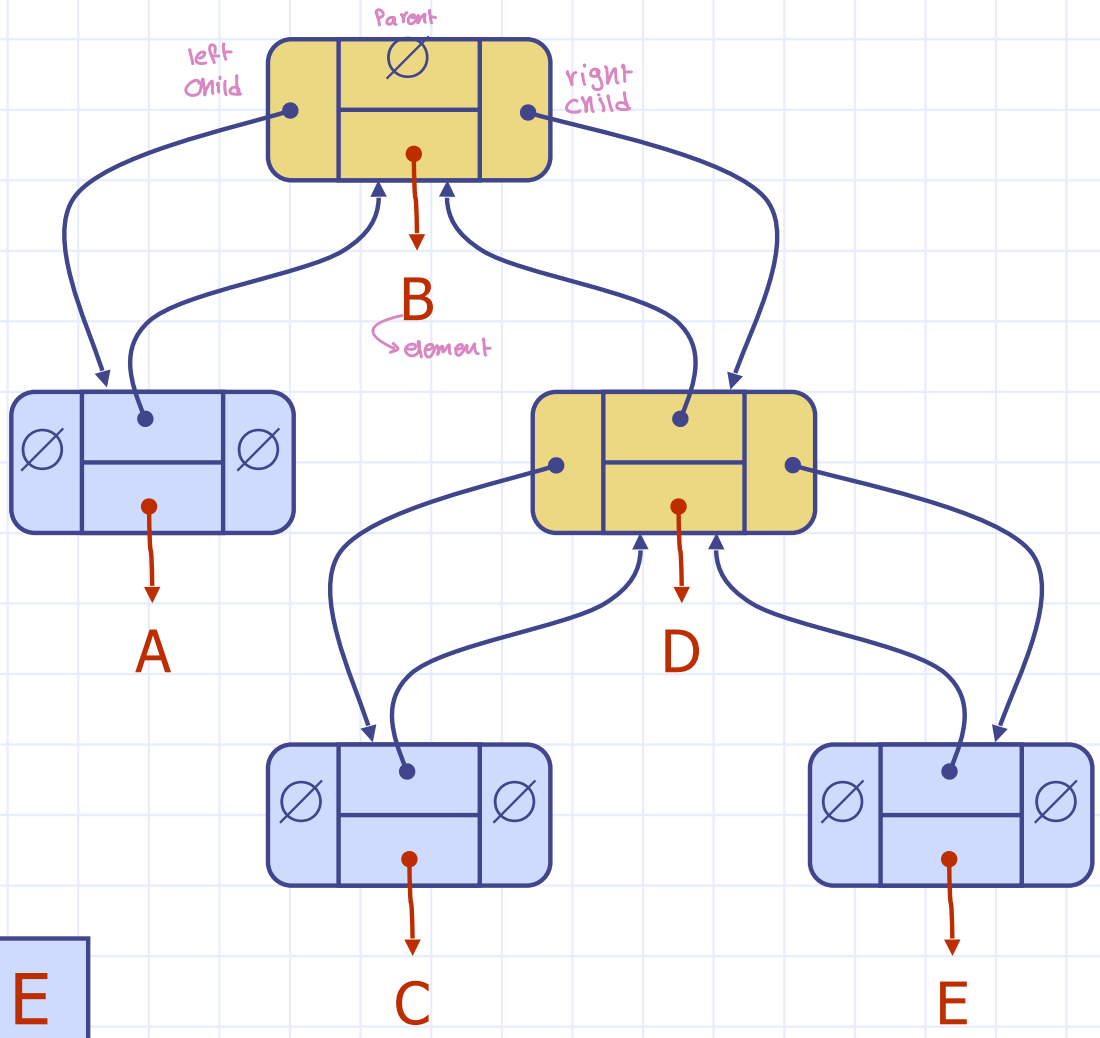

A
B
C
D


X
X
B
C
X
D

Linked Structure for Binary Trees

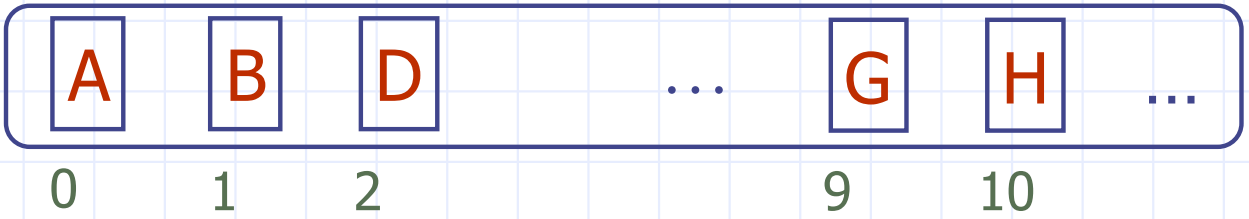
- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node
- Node objects implement the Position ADT

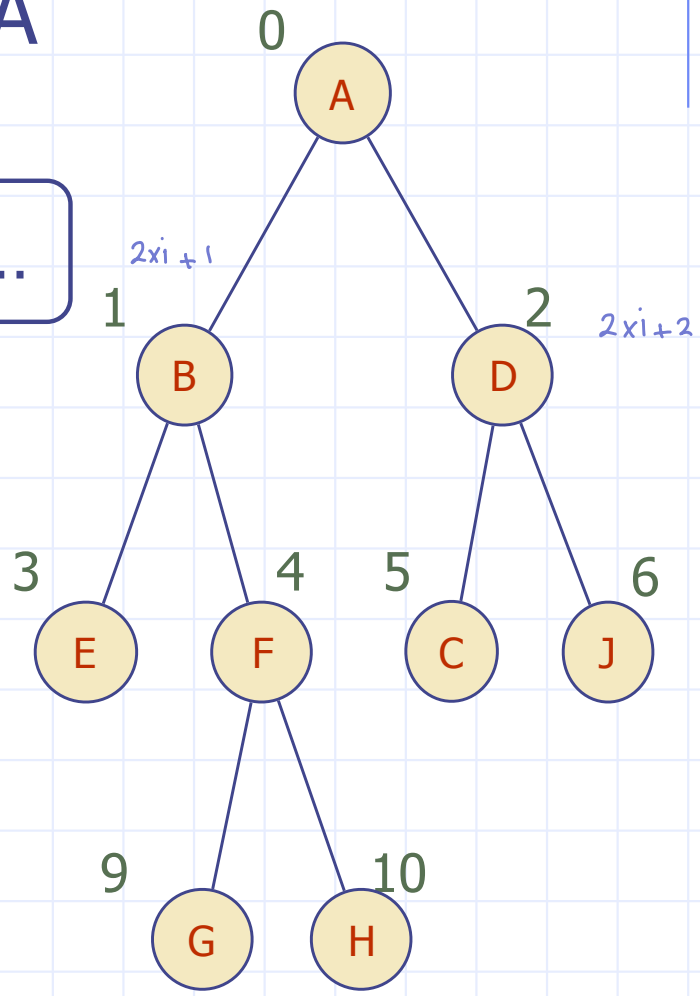




Array-Based Representation of Binary Trees

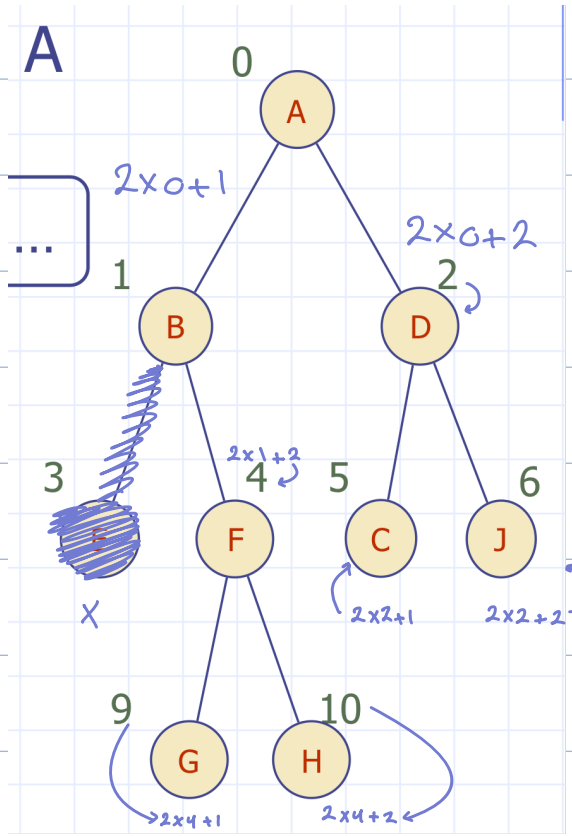
- Nodes are stored in an array A

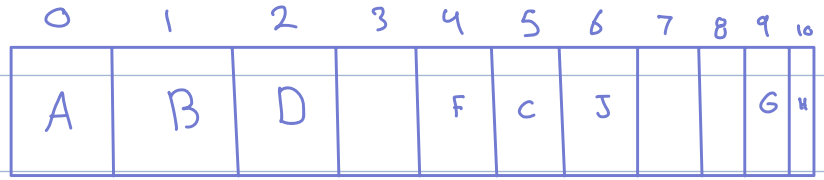




Node v is stored at A[rank(v)]

- rank(root) = 0
- if node is the left child of parent(node), rank(node) = 2 · rank(parent(node)) + 1
- if node is the right child of parent(node), rank(node) = 2 · rank(parent(node)) + 2


A
0
A
2x0+1
...
2x0+2
1
2
B
D
3
2x1+2
4
5
6
E
F
C
J
X
2x2+1
2x2+2
9
10
G
H
2x4+1
2x4+2


0 1 2 3 4 5 6 7 8 9 10
A
B
D
F
C
J
G

Complexity Linked Structure Tree Implementation

Operation	**Time**
size, isEmpty	$O(1)$
iterator, positions	$O(n)$
replace	$O(1)$
root, parent, children, left, right	$O(1)$
isInternal, isExternal, isRoot	$O(1)$