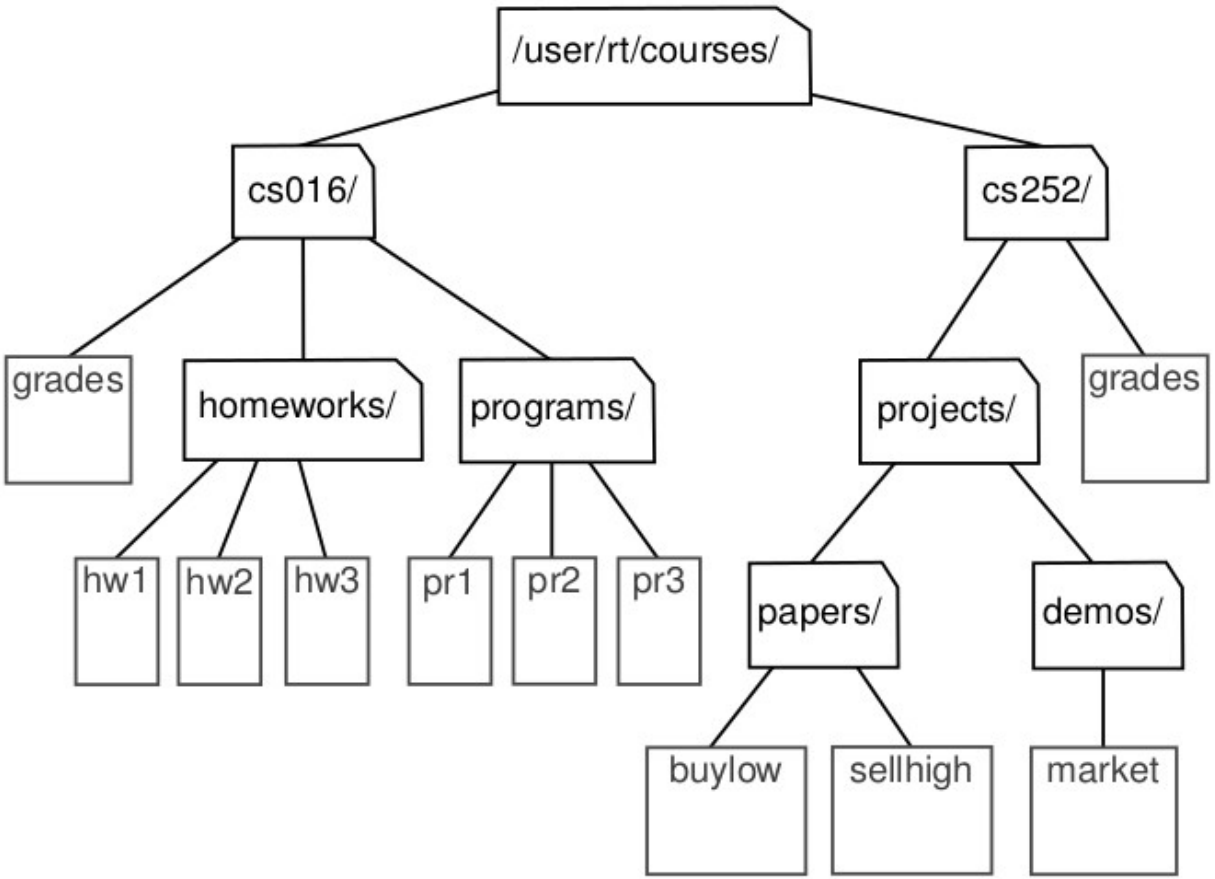
Trees

Applications – File directory



HTML & DOM tree

The browser parses the HTML and creates a tree of objects like this:

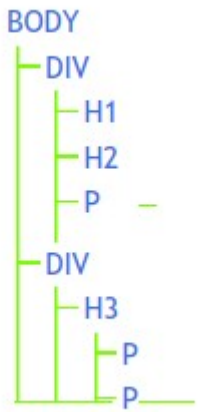
```
html
  head
    title
  body
    h1
    p
    script
```

```
<!doctype html>
<html>
  <head>
    <title>Hello, world!</title>
  </head>
  <body>
    <h1>Hello, world!</h1>
    <p>This is a hypertext document on the World Wide Web.</p>
    <script src="/script.js" async></script>
  </body>
</html>
```

```html
<!DOCTYPE html>
<html>
  <body>

      <div>
            <h1>DSA 2024</h1>
            <h2>Welcome to the class.</h2>
            <p> Well, actually not welcome as we have completed half way through </p>
      </div>

      <div>
            <h3> So far we have covered many data structures and ADTS
                  <p> today we discuss trees </p>
                  <p> yet to discuss are : priority queues, Hash maps etc </p>
            </h3>
      </div>
  </body>
</html>
```



DSA 2024

Welcome to the class.

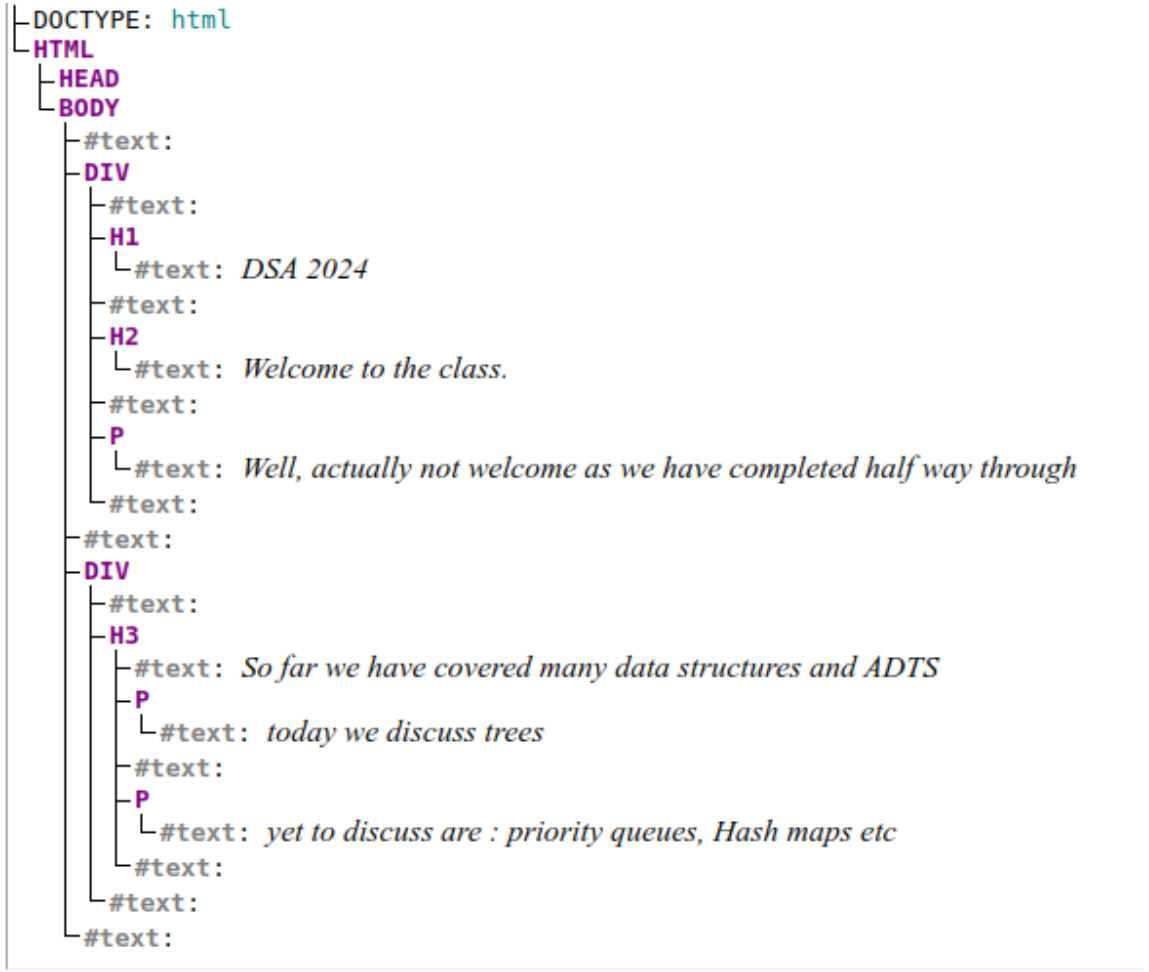
Well, actually not welcome as we have completed half way through

So far we have covered many data structures and ADTS

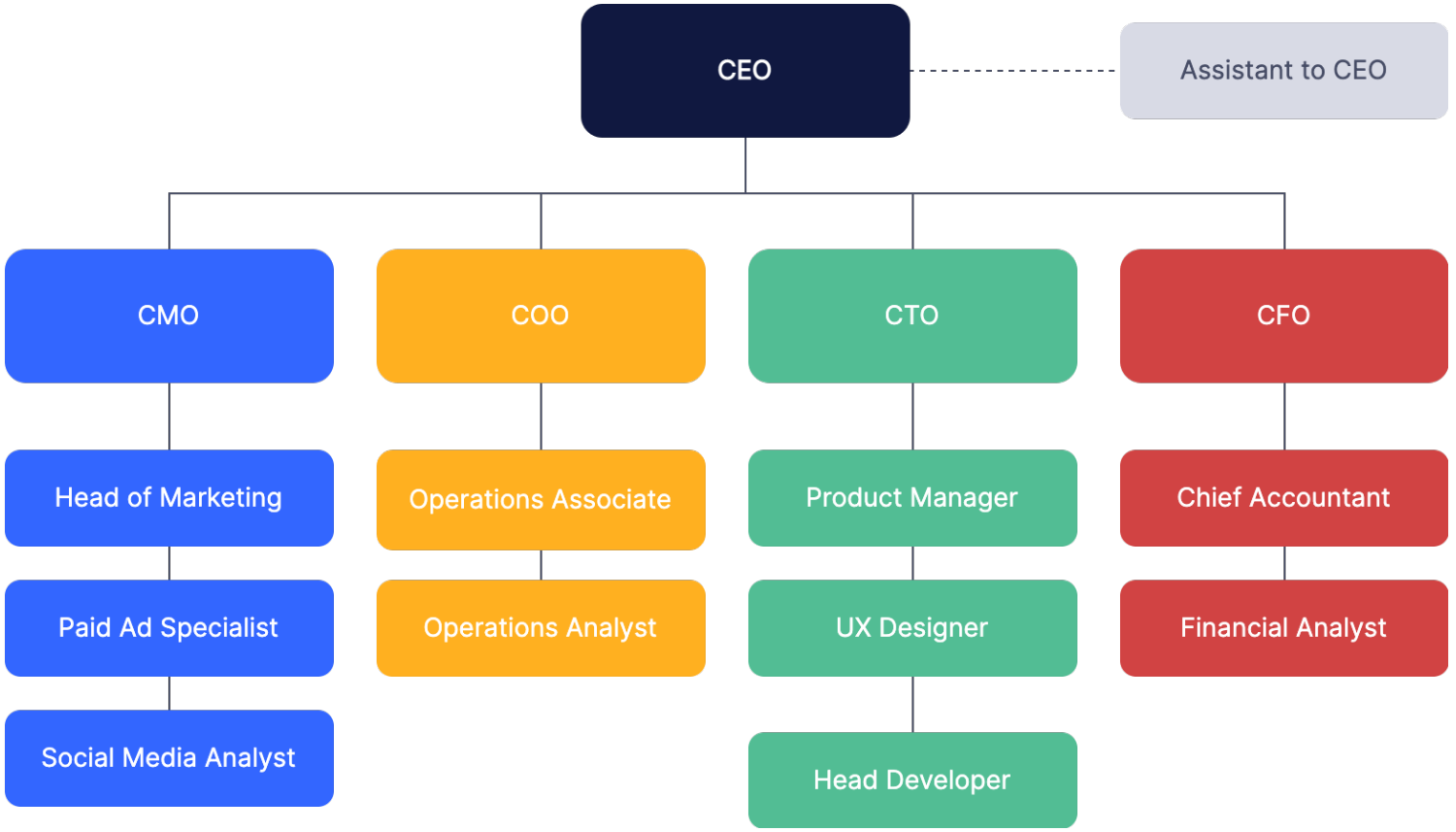
today we discuss trees

yet to discuss are : priority queues, Hash maps etc

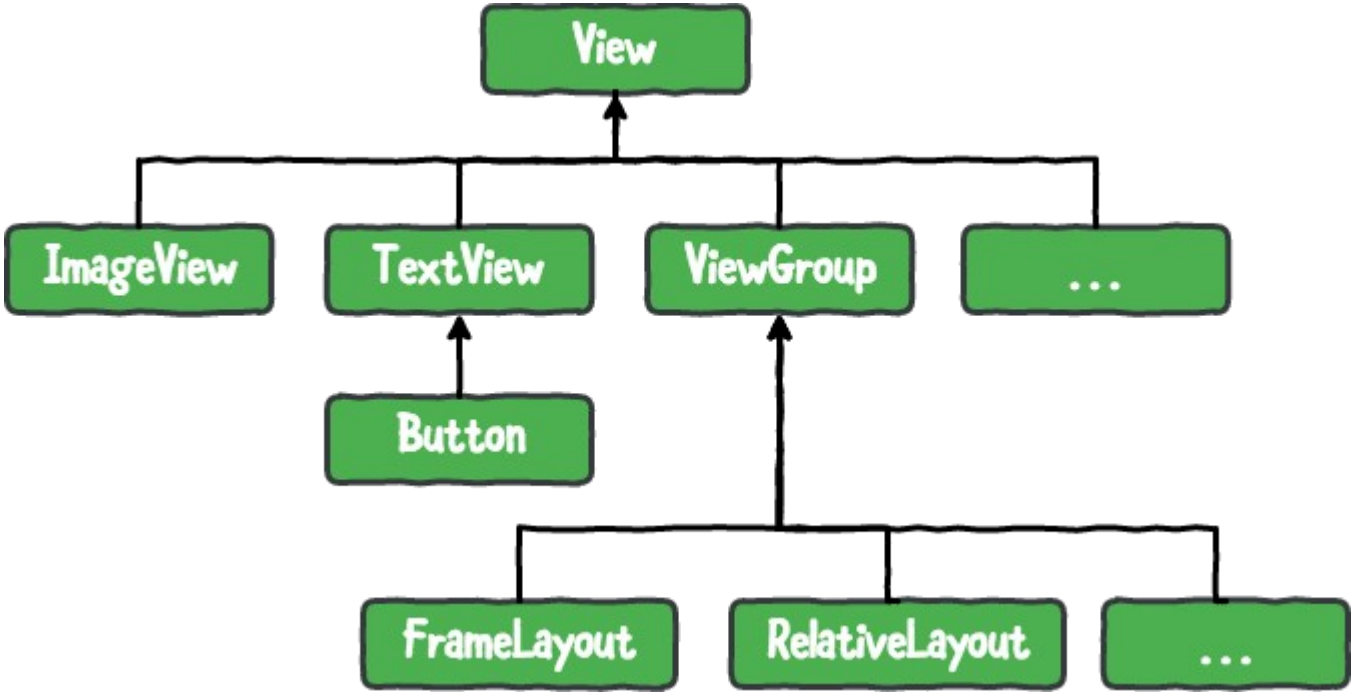
DOM tree


DOCTYPE: html
HTML
HEAD
BODY
#text:
DIV
#text:
H1
#text: DSA 2024
#text:
H2
#text: Welcome to the class.
#text:
P
#text: Well, actually not welcome as we have completed half way through
#text:
#text:
DIV
#text:
H3
#text: So far we have covered many data structures and ADTS
P
#text: today we discuss trees
#text:
P
#text: yet to discuss are : priority queues, Hash maps etc
#text:
#text:
#text:

Organisation structure

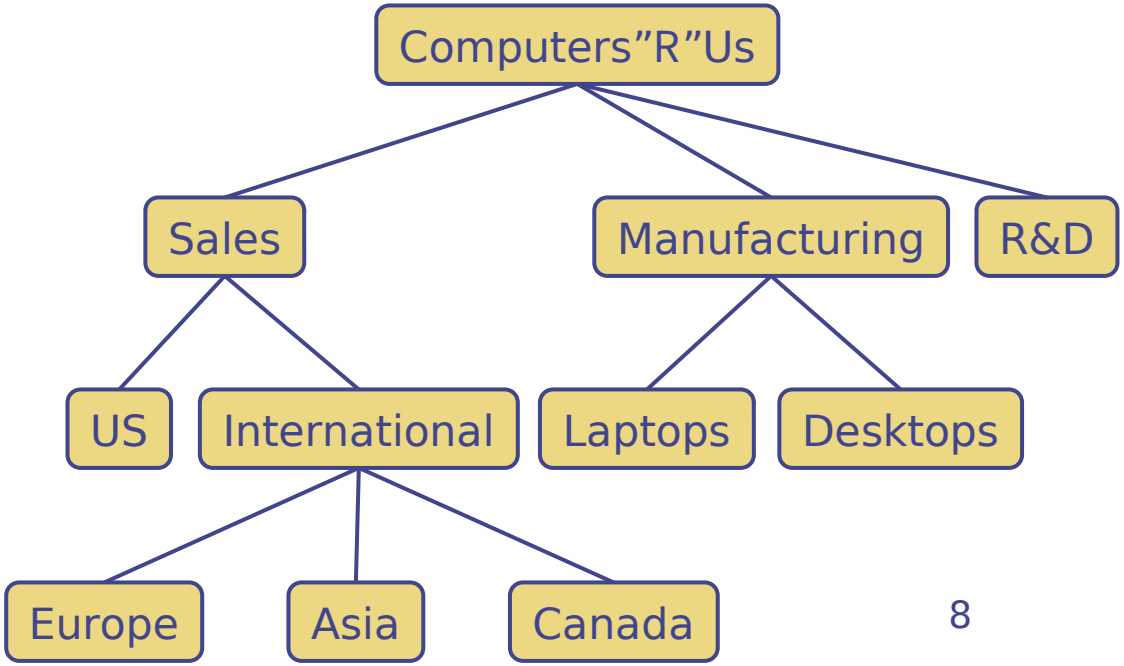


UI Layouts


View
ImageView
TextView
ViewGroup
...
Button
FrameLayout
RelativeLayout
...

What is a Tree?

- Tree is used to represent hierarchical relationships between elements. It consists of a collection of nodes connected by edges signifying parent-child relationship.
- A node can have zero or more children. Each node has at most one parent node except the root node which has none.

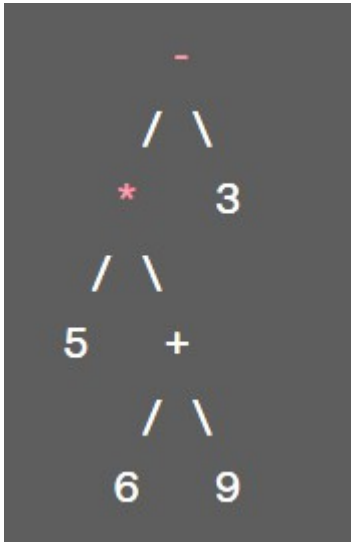


Applications summary

- File Systems
- Family Trees
- Social Networks
- Syntax Trees in Compilers
- UI layouts
- DOM trees to represent webpages
- Decision Trees in Machine Learning

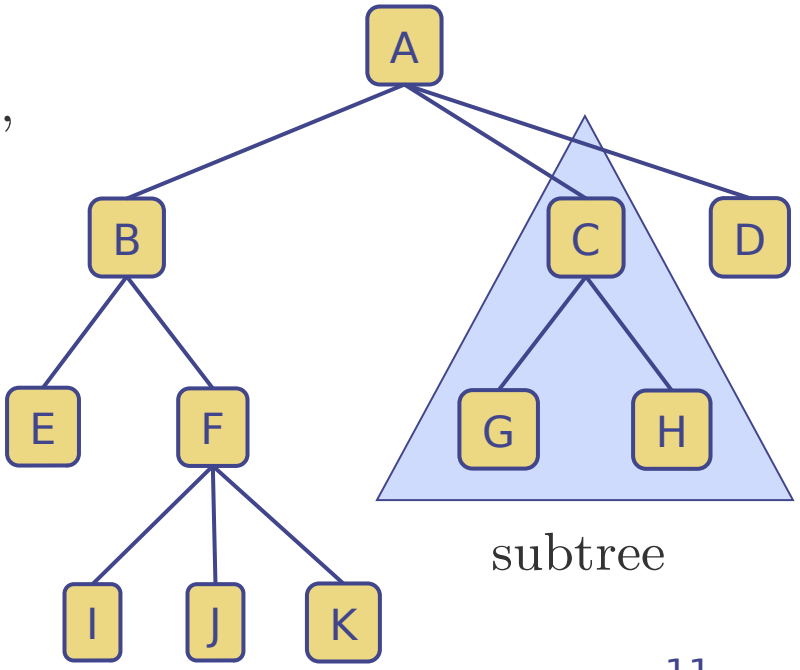
Example of a Syntax tree

$5 \times (6+9) - 3$:



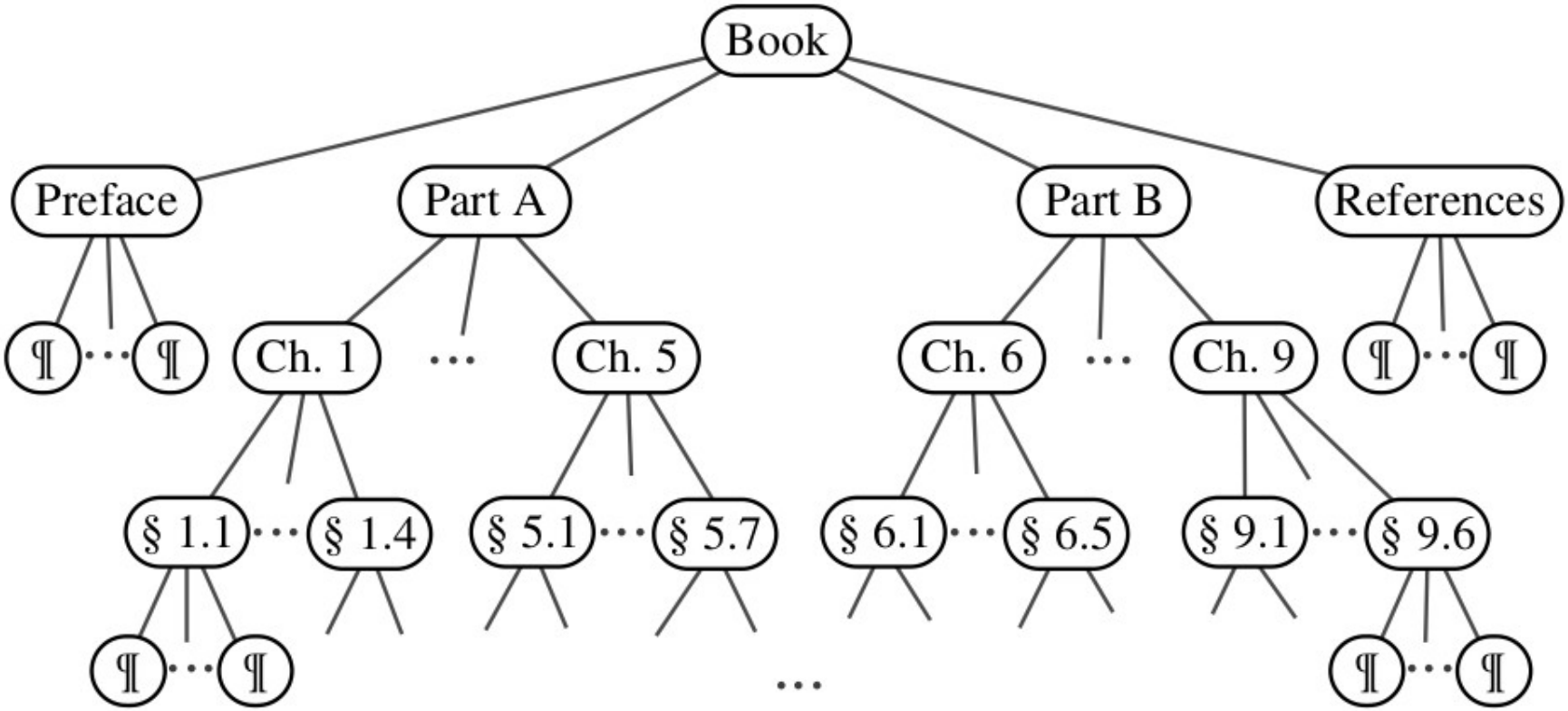
Tree Terminology

- 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 among its leaf node (3)
- Descendant of a node: child, grandchild, grand-grandchild, etc.
- Subtree: tree consisting of a node and its descendants

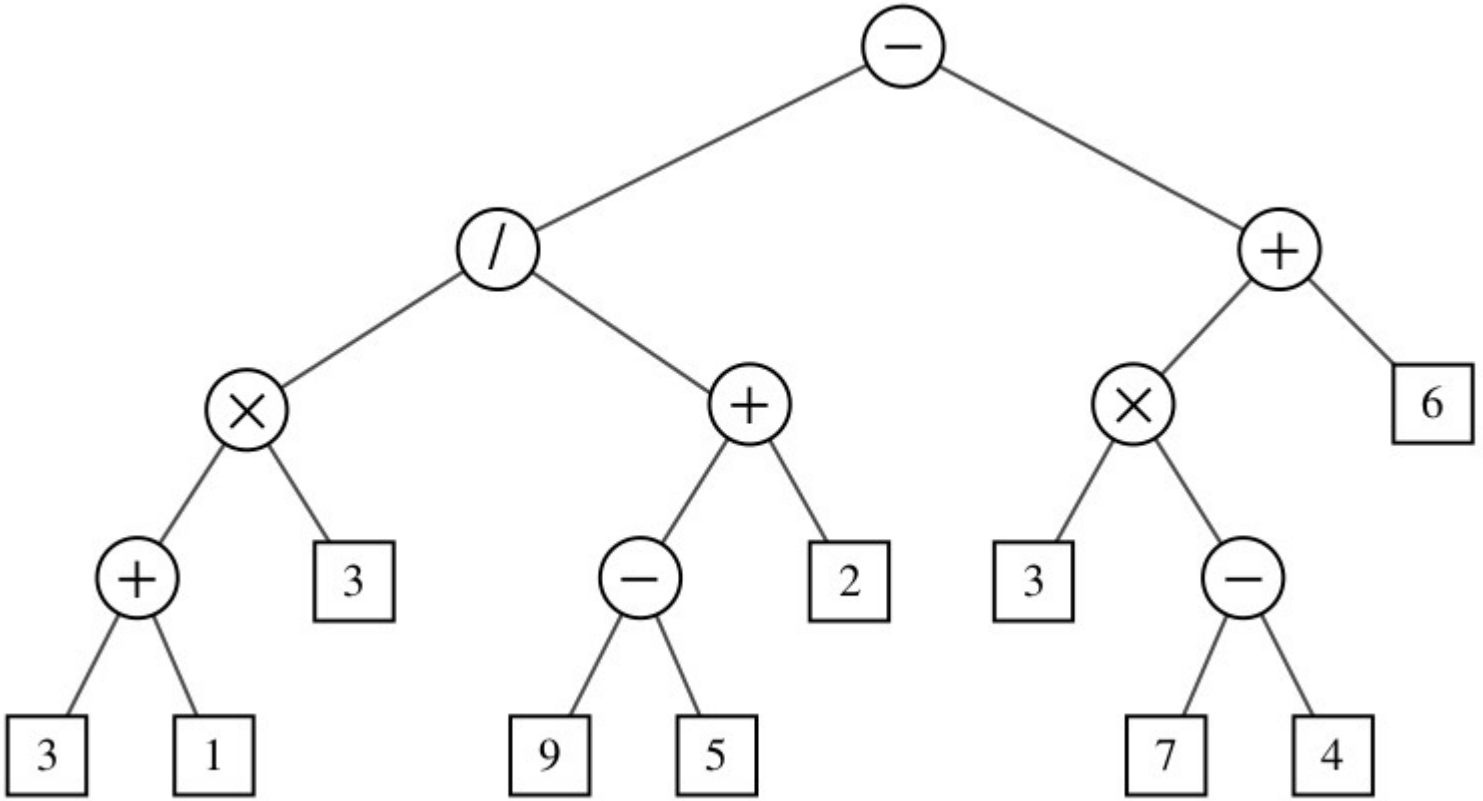


subtree

Ordered Tree example



Binary Tree



Tree ADT

A tree ADT is generally defined using the concept of a position as an abstraction for a node of a tree.

A position object for a tree supports the method:

p.element( ): Return the element stored at position p

Generic methods:

Integer len()

Boolean is_empty()

Iterator positions()

Iterator iter()

Accessor methods:

position root()

position parent(p)

Iterator children(p)

Integer num_children(p)

Tree ADT

Query methods:

Boolean is_leaf(p)

Boolean is_root(p)

Update method:

element replace (p, o)

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

Abstract Tree Class in Python

```
1  class Tree:
2    """Abstract base class representing a tree structure."""
3
4    #-------------------------- nested Position class --------------------------
5    class Position:
6      """An abstraction representing the location of a single element."""
7
8      def element(self):
9        """Return the element stored at this Position."""
10       raise NotImplementedError('must be implemented by subclass')
11
12     def __eq__(self, other):
13       """Return True if other Position represents the same location."""
14       raise NotImplementedError('must be implemented by subclass')
15
16     def __ne__(self, other):
17       """Return True if other does not represent the same location."""
18       return not (self == other)           # opposite of __eq__
19
```

Abstract Tree Class in Python

```
20    # ---------- abstract methods that concrete subclass must support ----------
21    def root(self):
22      """Return Position representing the tree's root (or None if empty)."""
23      raise NotImplementedError('must be implemented by subclass')
24
25    def parent(self, p):
26      """Return Position representing p's parent (or None if p is root)."""
27      raise NotImplementedError('must be implemented by subclass')
28
29    def num_children(self, p):
30      """Return the number of children that Position p has."""
31      raise NotImplementedError('must be implemented by subclass')
32
33    def children(self, p):
34      """Generate an iteration of Positions representing p's children."""
35      raise NotImplementedError('must be implemented by subclass')
36
37    def __len__(self):
38      """Return the total number of elements in the tree."""
39      raise NotImplementedError('must be implemented by subclass')
```

Abstract Tree Class in Python

```
40    # ---------- concrete methods implemented in this class ----------
41    def is_root(self, p):
42      """Return True if Position p represents the root of the tree."""
43      return self.root( ) == p
44
45    def is_leaf(self, p):
46      """Return True if Position p does not have any children."""
47      return self.num_children(p) == 0
48
49    def is_empty(self):
50      """Return True if the tree is empty."""
51      return len(self) == 0
```

Tree Traversal Algorithms

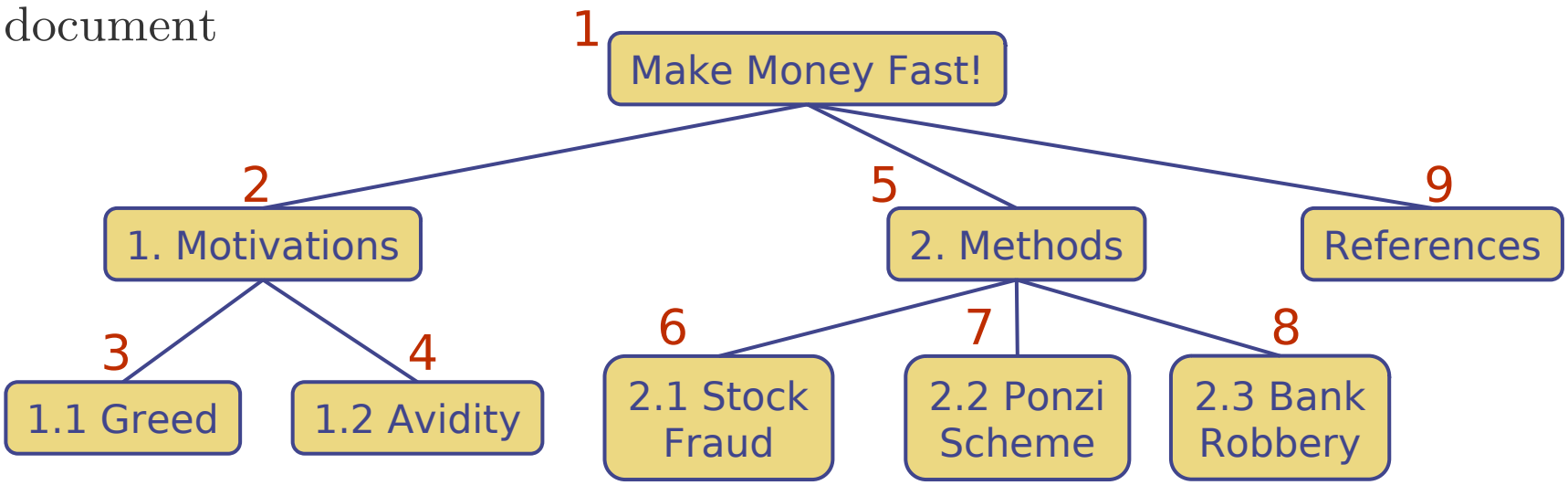
Preorder Traversal

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*)

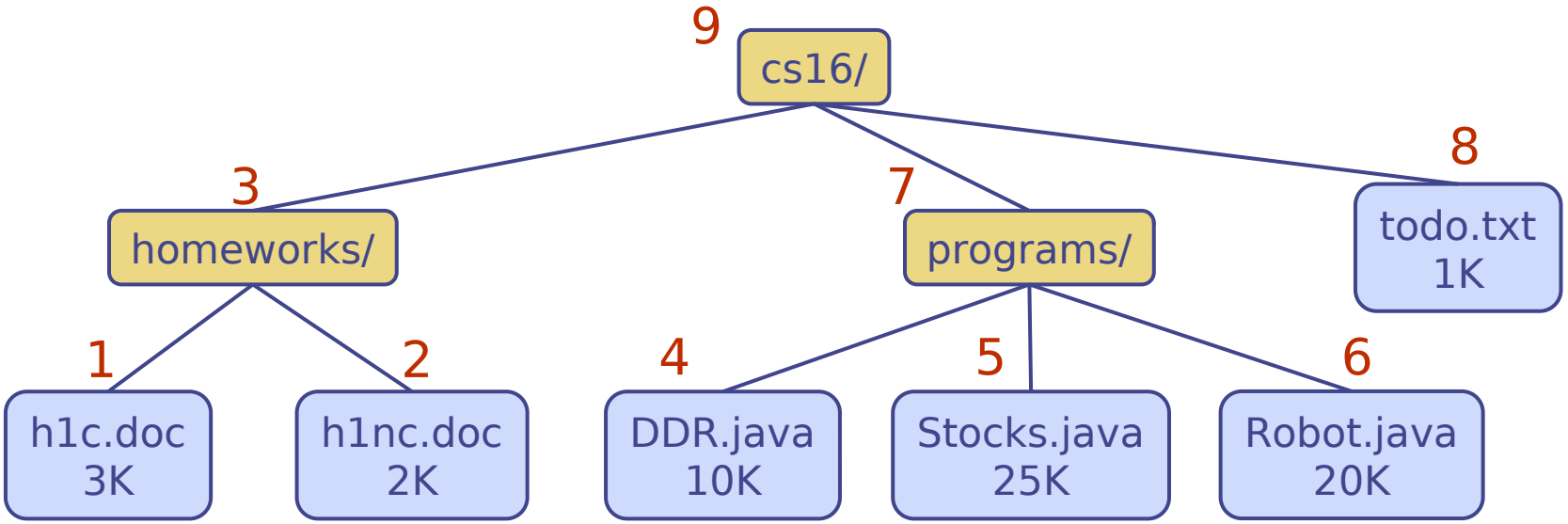


Postorder Traversal

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)



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

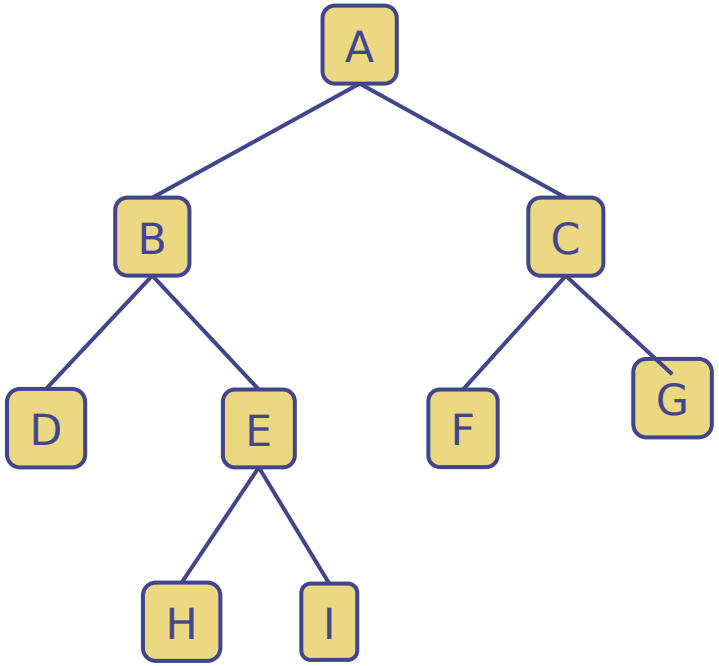
We call the children of an internal node left child and right child

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



Binary Trees Properties

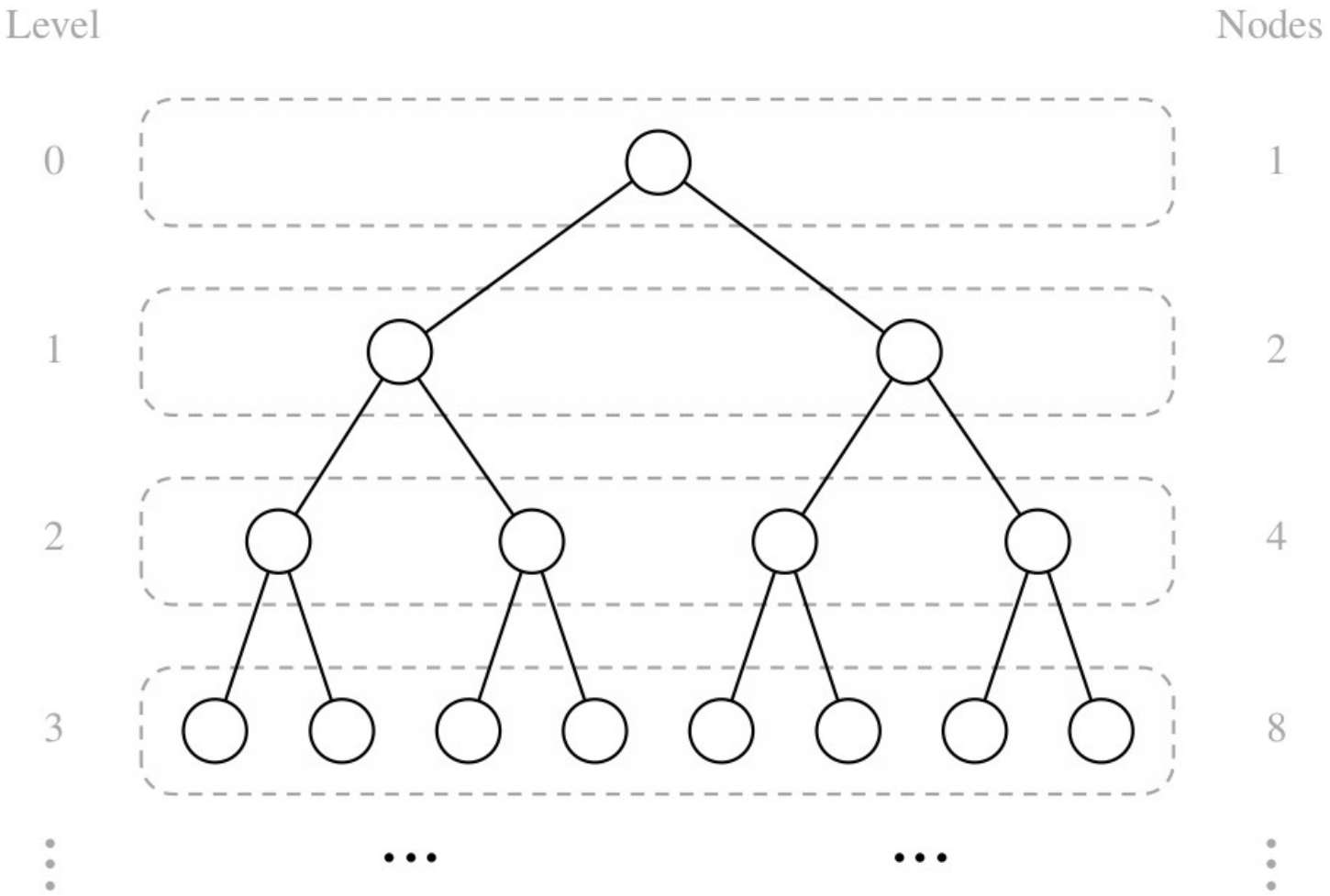
Proposition 8.8: *Let T be a nonempty binary tree, and let n, n_E, n_I and h denote the number of nodes, number of external nodes, number of internal nodes, and height of T, respectively. Then T has the following properties:*

1. $h+1 \leq n \leq 2^{h+1}-1$
2. $1 \leq n_E \leq 2^h$
3. $h \leq n_I \leq 2^h-1$
4. $\log(n+1)-1 \leq h \leq n-1$

Also, if T is proper, then T has the following properties:

1. $2h+1 \leq n \leq 2^{h+1}-1$
2. $h+1 \leq n_E \leq 2^h$
3. $h \leq n_I \leq 2^h-1$
4. $\log(n+1)-1 \leq h \leq (n-1)/2$

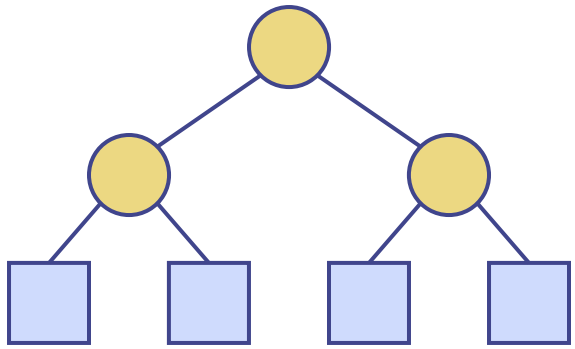
Proper/Full Binary Tree

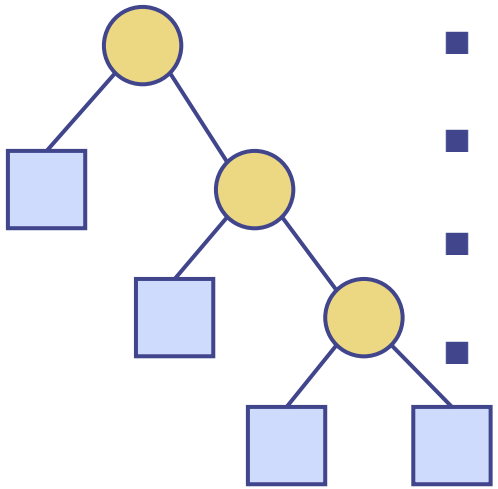


Properties of Proper Binary Trees

Notation

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





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

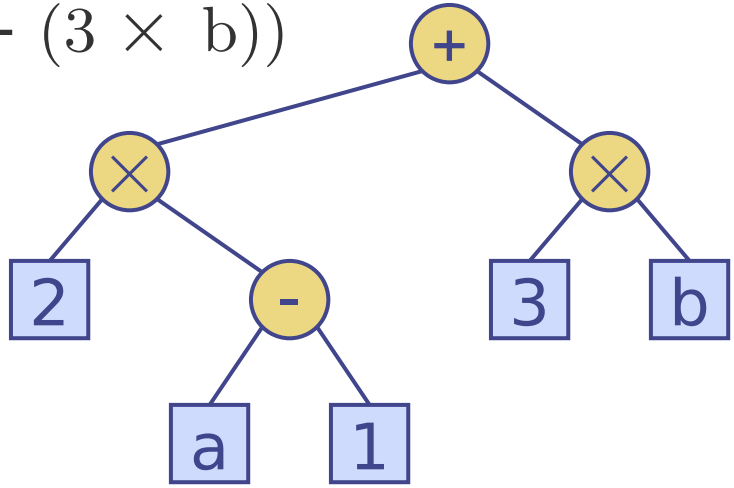
Arithmetic Expression Tree

Binary tree associated with an arithmetic expression

internal nodes: operators

external nodes: operands

Example: arithmetic expression tree for the expression $(2 \times (a - 1) + (3 \times b))$



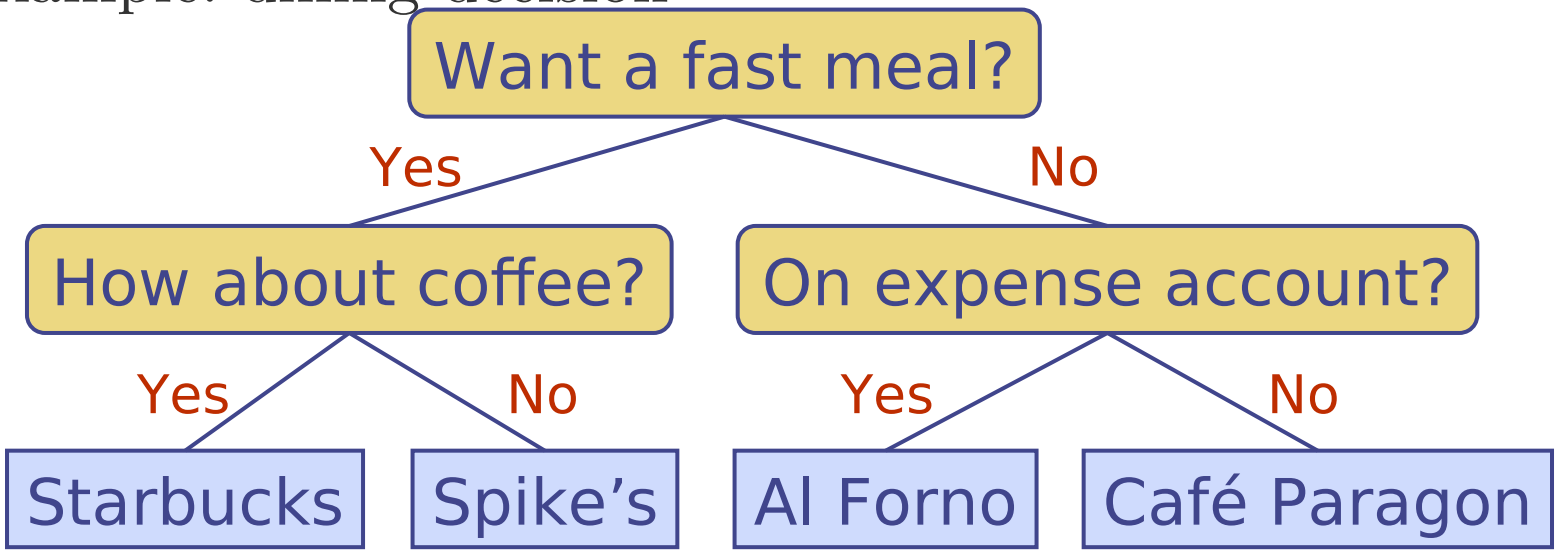
Decision Tree

Binary tree associated with a decision process

internal nodes: questions with yes/no answer

external nodes: decisions

Example: dining decision



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)

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)*

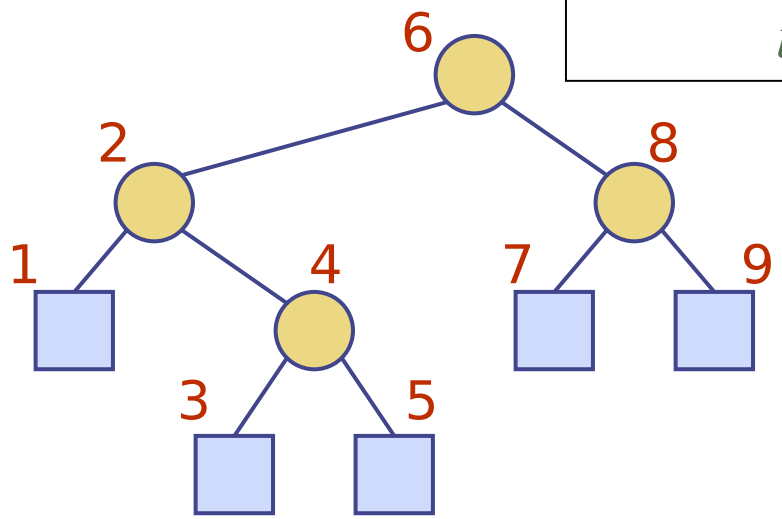
if *v* has a left child

***inOrder* (*left* (*v*))**

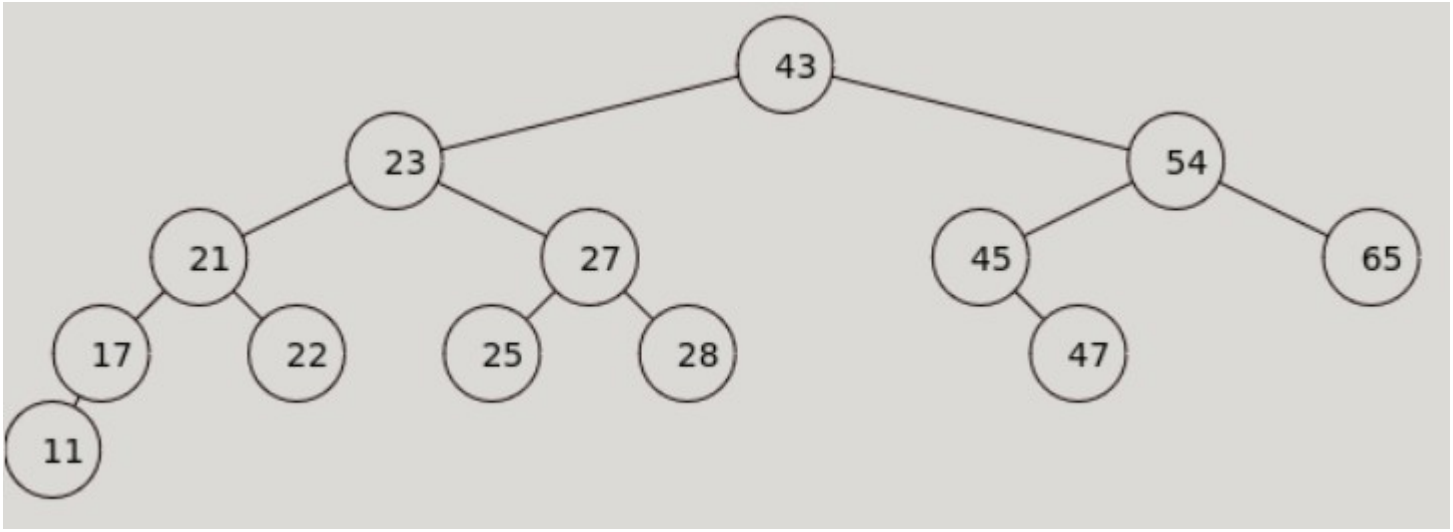
visit(v)

if *v* has a right child

***inOrder* (*right* (*v*))**



Example – display elements using traversals



Preorder: 43 23 21 17 11 22 27 25 28 54 45 47 65

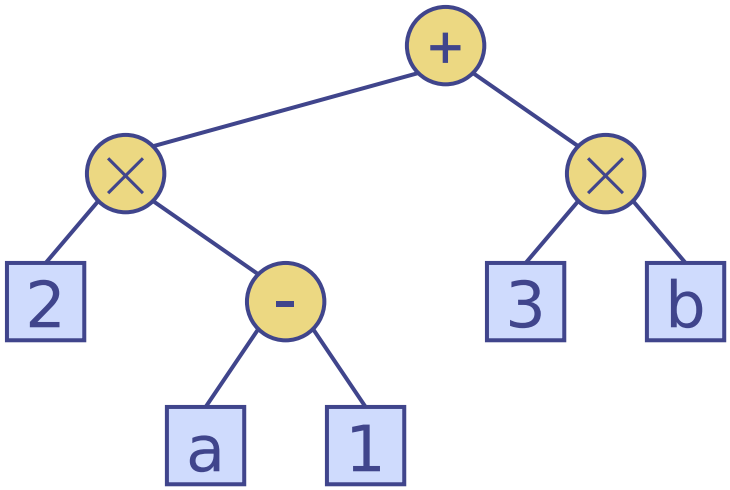
Postorder: 11 17 22 21 25 28 27 23 47 45 65 54 43

Inorder: 11 17 21 22 23 25 27 28 43 45 47 54 65

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 (")")

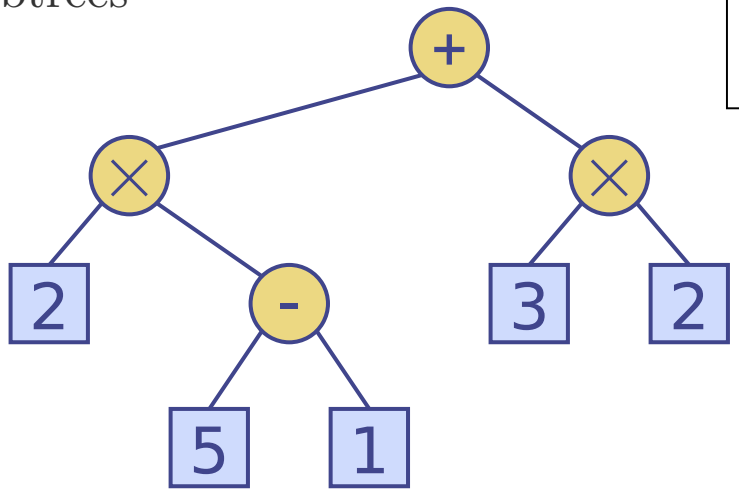
((2 × (a - 1)) + (3 × b))

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



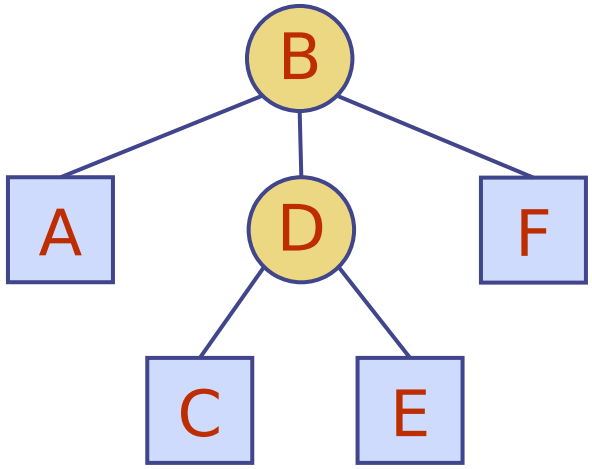
Algorithm *evalExpr*(*v*)
  if *is_leaf* (*v*)
    return *v.element* ()
  else
    $x \leftarrow$ *evalExpr*(*left* (*v*))
    $y \leftarrow$ *evalExpr*(*right* (*v*))
    $\Diamond \leftarrow$ operator stored at *v*
    return $x \Diamond y$

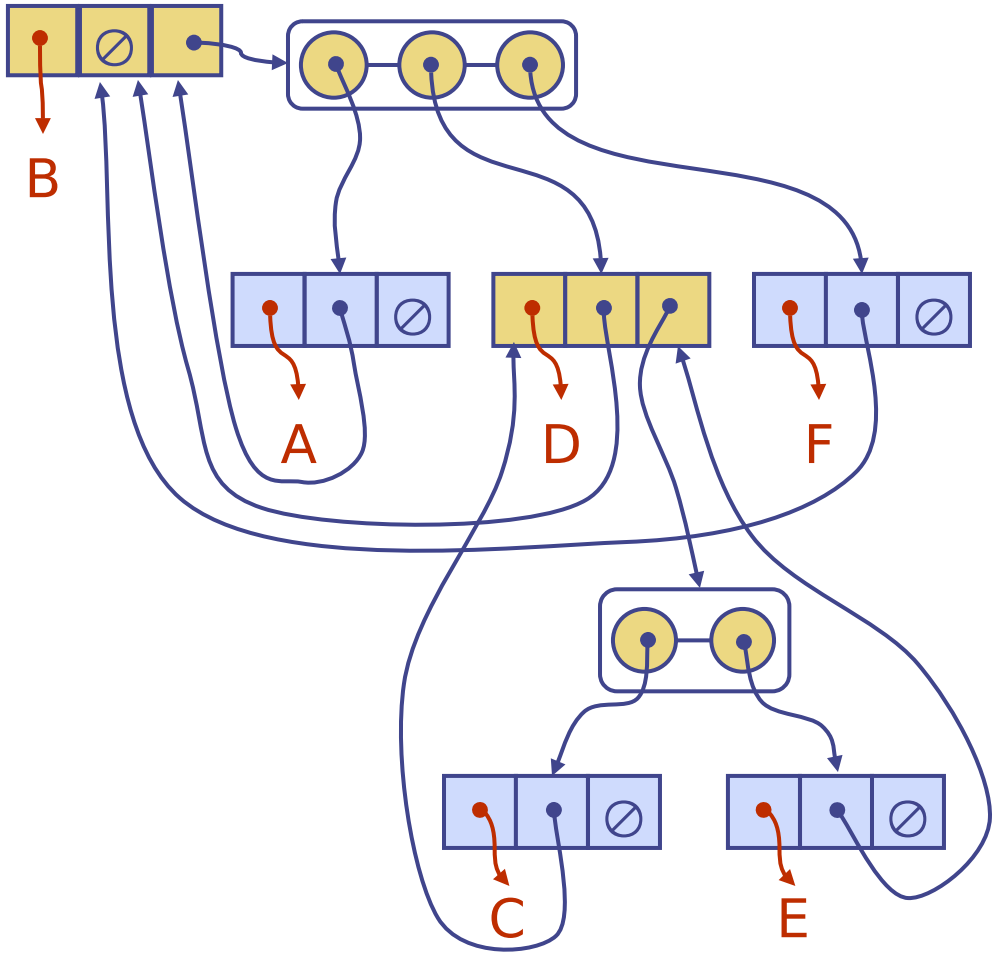
Linked Structure for Trees

A node is represented by an object storing

- Element
- Parent node
- Sequence of children nodes

Node objects implement the Position ADT



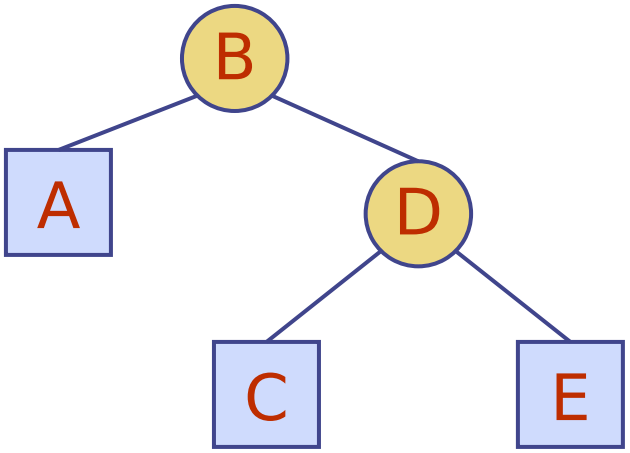


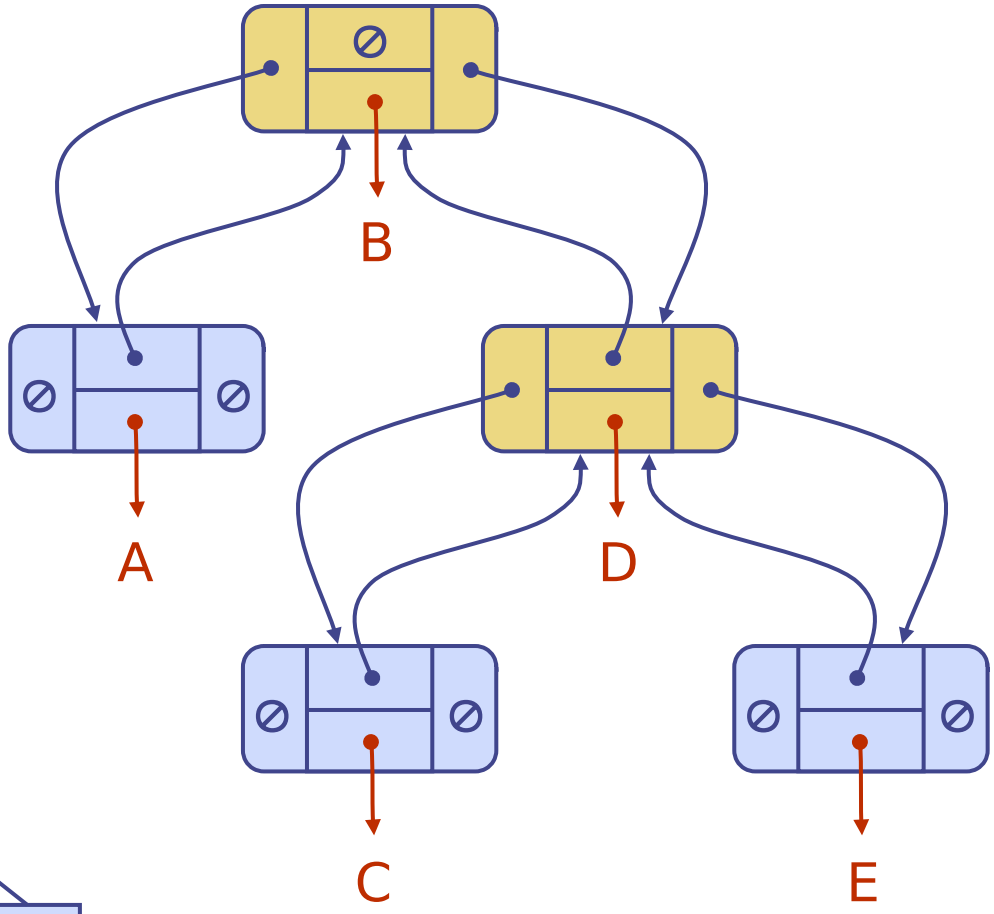
Linked Structure for Binary Trees

A node is represented by an object storing

- Element
- Parent node
- Left child node
- Right child node

Node objects implement the Position ADT

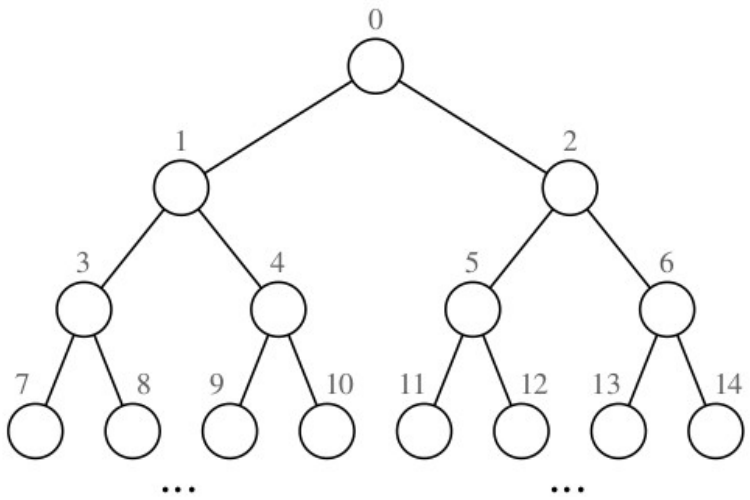




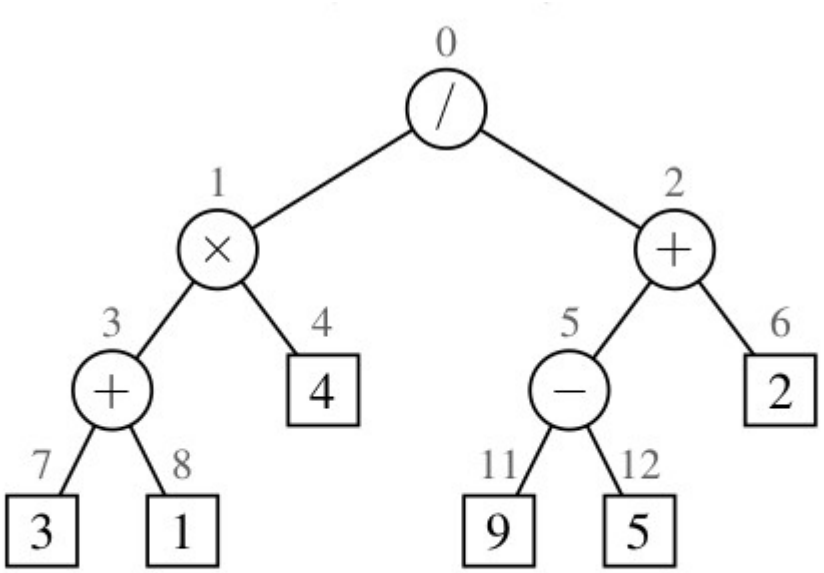
Array-Based Representation of a Binary Tree

- If p is the root of T , then f (p) = 0.
- If p is the left child of position q, then f (p) = 2 f (q) + 1.
- If p is the right child of position q, then f (p) = 2 f (q) + 2.

The numbering function **f** is known as a **level numbering** of the positions in a binary tree T , for it numbers the positions on each level of T in increasing order from left to right.

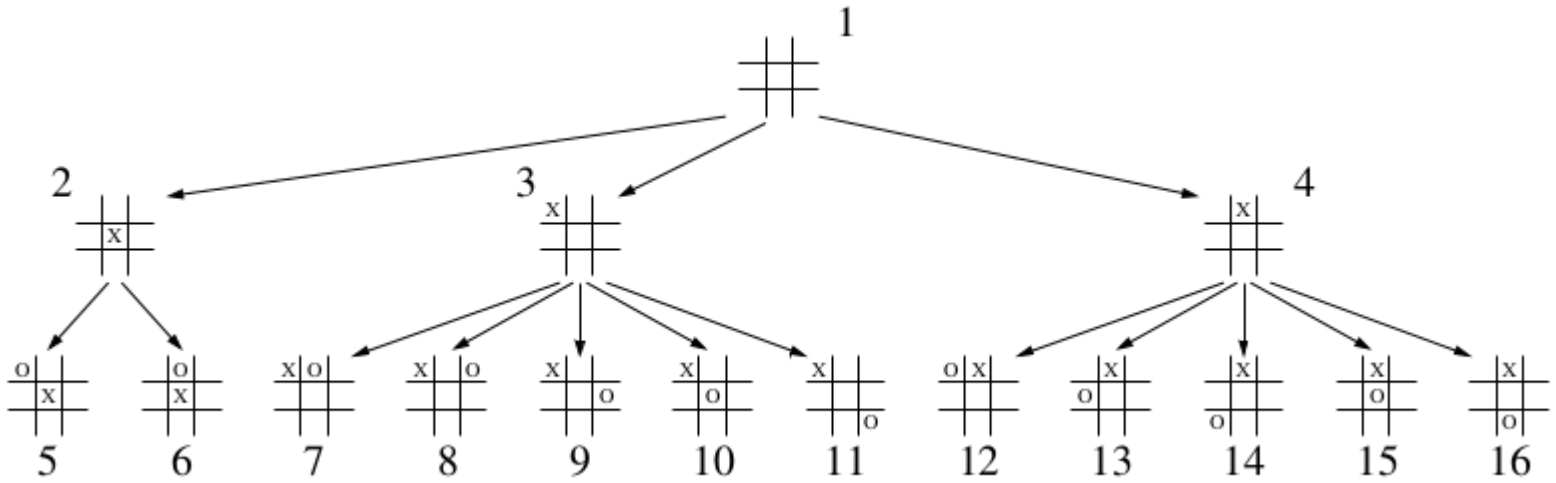


Array-Based Representation of a Binary Tree



/	×	+	+	4	−	2	3	1			9	5		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

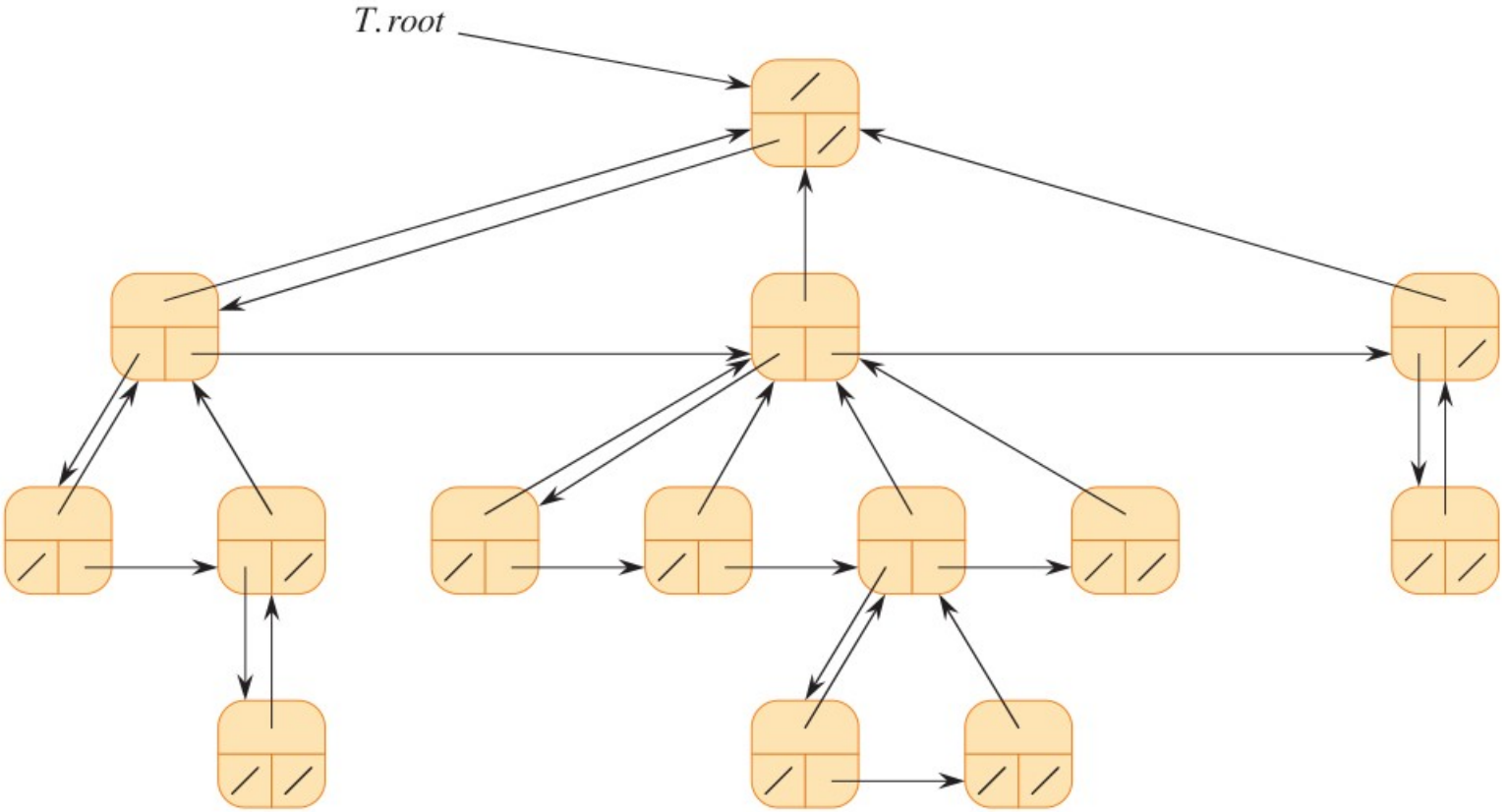
Breadth-First Tree Traversal



Breadth-First Tree Traversal

```
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}
```

left-child, right-sibling representation of a tree



A scheme to represent trees with arbitrary numbers of children.

About yield statement

```
def even_numbers():
  i = 0
  while True:
    yield i
    i += 2

# print the first 6 even numbers
for n in even_numbers():
  if n > 10:
    break
  print(n)
```

Memory Efficiency, Lazy Evaluation, can be used as a iterator (without any index)

References

Data Structures and Algorithms in Python

Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser

Introduction to Algorithms

Leiserson,Stein,Rivest,Cormen

Algorithms, 4th Edition

Robert Sedgewick and Kevin Wayne

Few Images from the internet