Data structures III – Trees

Recursion Refresher - 1

- For some problems, it may be natural to define the problem in terms of the problem itself
- Recursion means that a function calls itself
- Useful for problems that can be represented by a simpler version of the same problem
- Example: the factorial function

$$6! = 6 * 5 * 4 * 3 * 2 * 1$$

- Can be written as:

$$6! = 6 * 5!$$

- Thus factorial function can be expressed as:

$$n! = n * (n-1)!$$

Recursion Refresher - 2

Iterative Solution

```
Public int fac(int numb){
    int product=1;
    while(numb>1){
       product *= numb;
       numb--;
    }
    return product;
}
```

Recursive Solution

```
public int fac(int numb){
  if(numb<=1)
    return 1;
  else
    return  numb*fac(numb-1);
}
```

Recursion Refresher - 3

Exponent Function

```
int exp(int num, int power)
{
   if(power ==0)
        return 1;

   return num*exp(num, power-1);
}
```

General Form

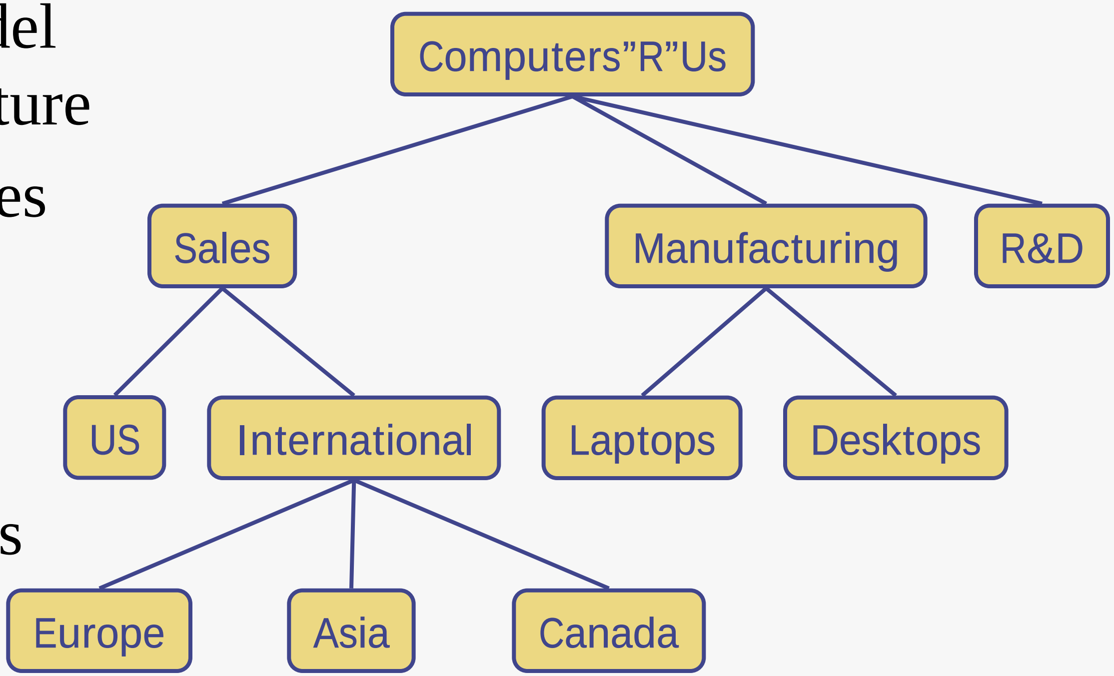
```
int recur_fn(param)
{
   if(stopping condition)
        return stopping value;

   return function of
   recur_fn(revised param);
}
```

TREES

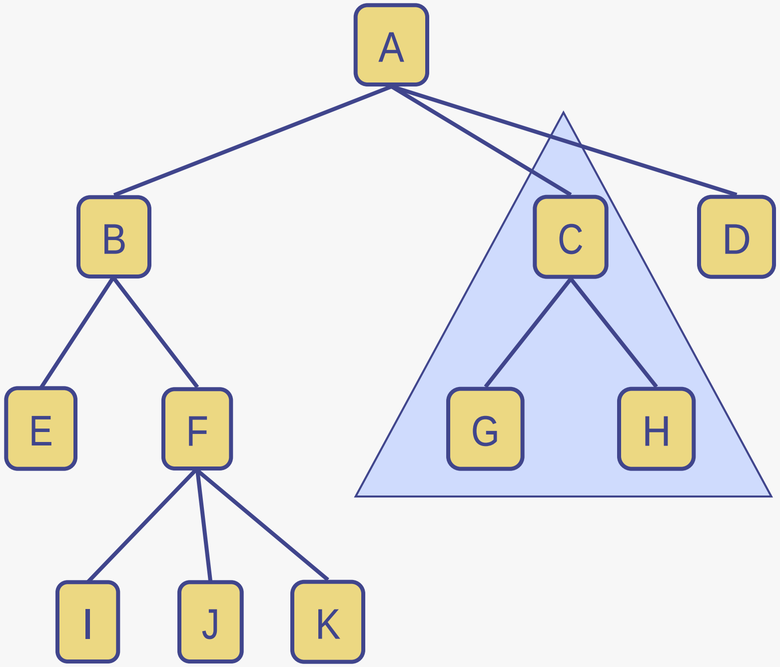
What is a Tree?

- 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



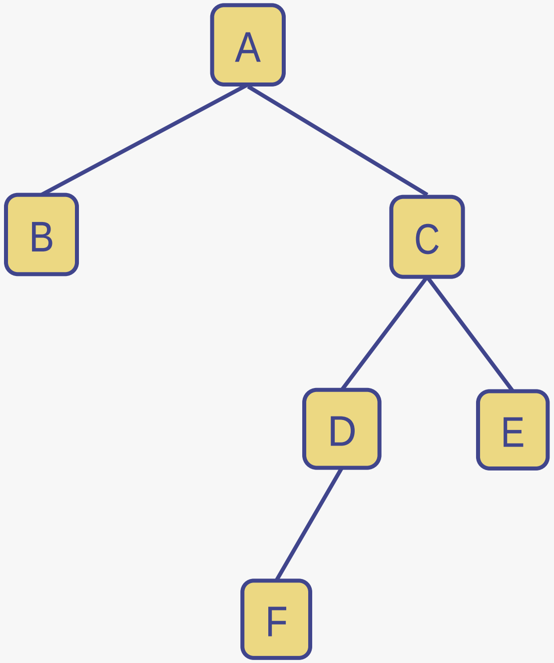
Tree Terminology

- **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. (B is grandparent of I)
- **Descendant** of a node: child, grandchild, grand-grandchild, etc.
- **Depth** of a node: number of ancestors (depth of E is 2)
- **Height** of a tree: maximum depth of any node (3)
- **Subtree**: tree consisting of a node and its descendants



Traversing a Tree

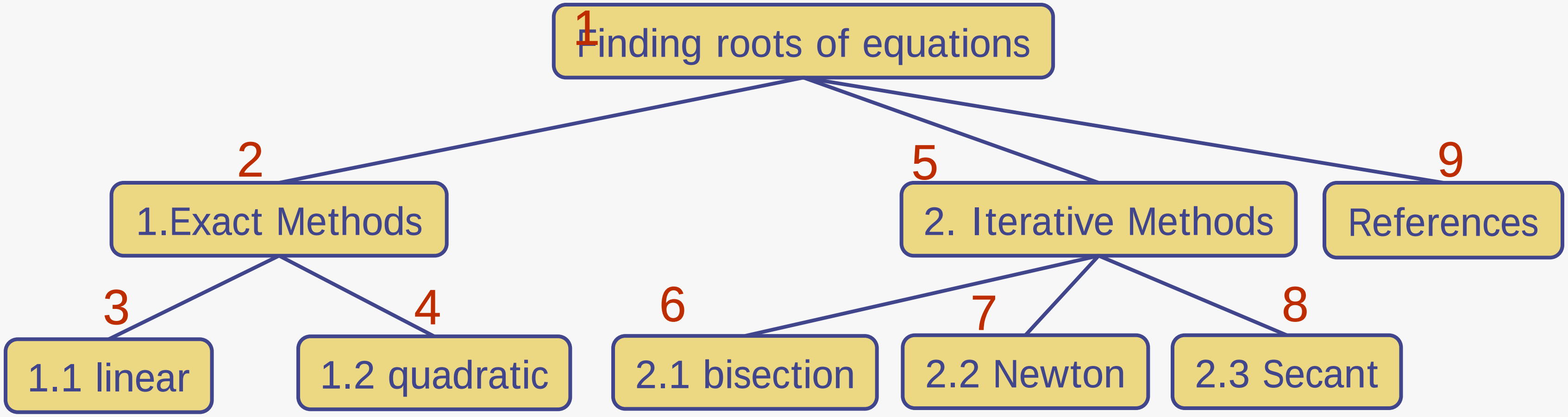
- Trees can be traversed in three orders
- Order chosen depends on
  - What the tree is being used for
  - What the traversal is supposed to accomplish
- Traversal is done **recursively!**
  - Treat L, R as trees in their own right
  - Recursively visit them
- **Pre-order**: root, L, R
  - A, B, C, D, F, E
- **In-order**: L, root, R
  - B, A, F, D, C, E
- **Post-order**: L, R, root
  - B, F, D, E, C, A



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, e.g. Table of Contents

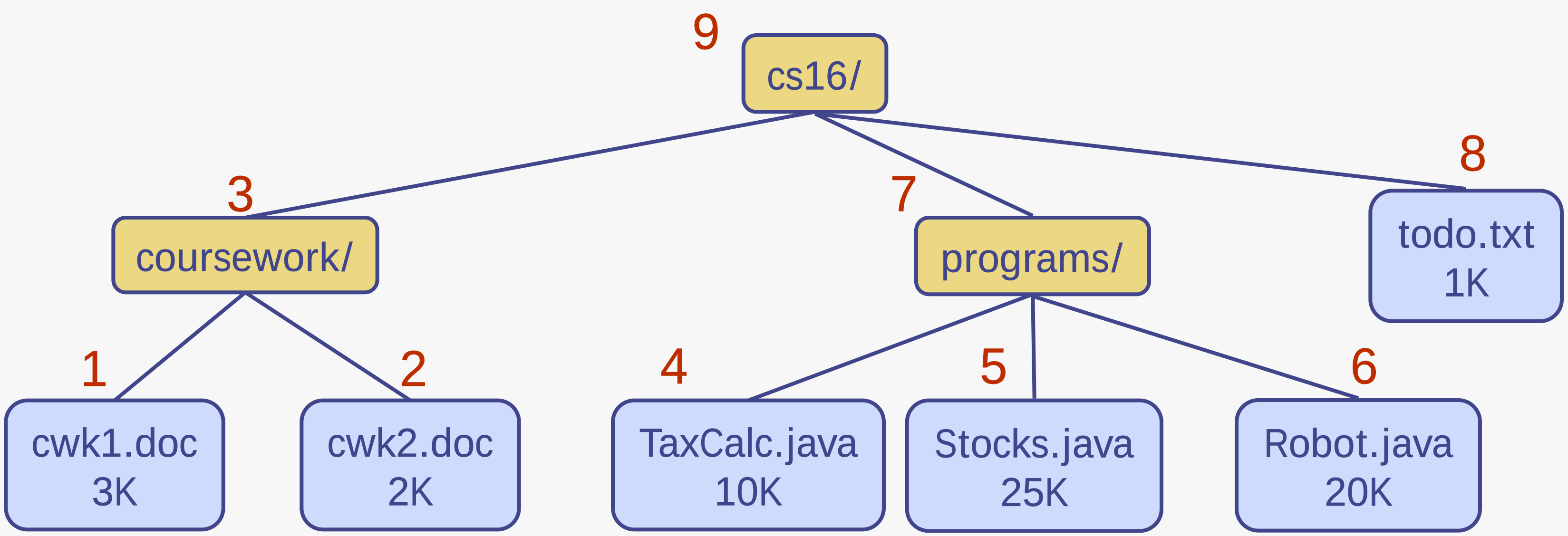
Algorithm *preOrder*(v)
    visit(v)
    for each child w of v
        preOrder (w)



Postorder Traversal

- Visits nodes in this order
- In a postorder traversal, **a node is visited after its descendants**
- Application: disk space used by files in a directory and its subdirectories

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

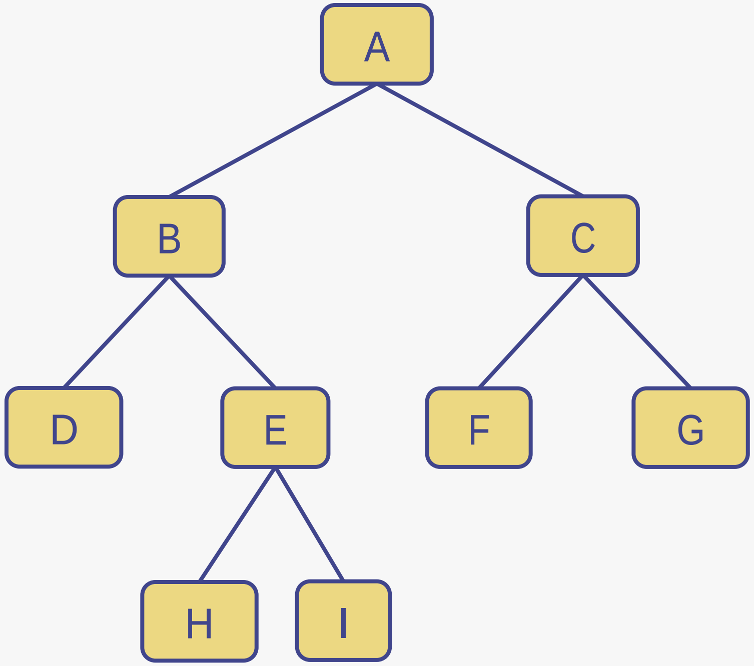


Binary Trees

- A binary tree is a tree with the following properties:

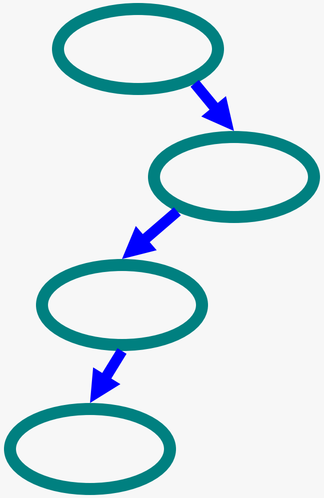
  - Each internal node has at most two children (exactly two for ***proper*** binary trees)
  - The children of a node are an ordered pair

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

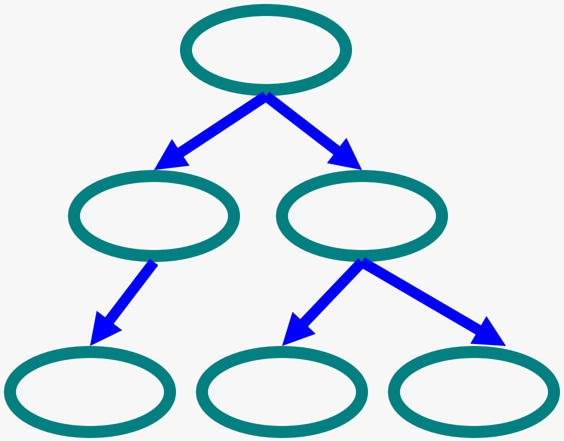


Types of Binary Trees

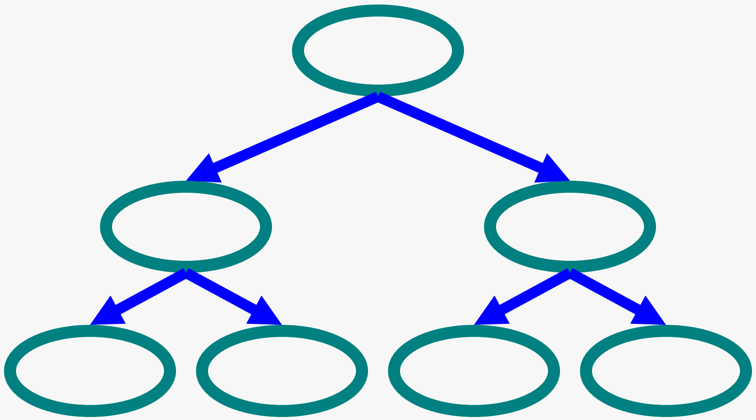
- Degenerate – only one child
- Balanced – mostly two children
- Proper – always two children



Degenerate binary tree



Balanced binary tree

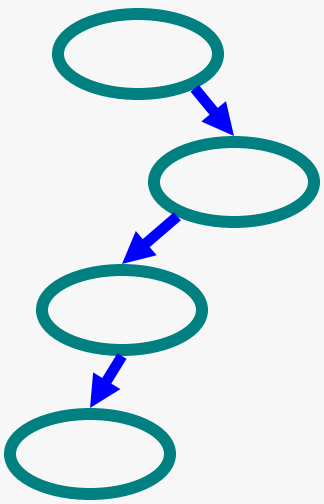


Proper binary tree

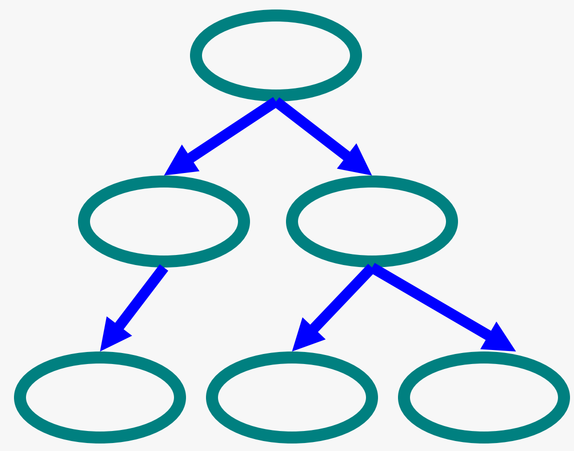
Binary Trees Properties

- ***Degenerate***
  - Height = O(n) for n nodes
  - Similar to linear list

- ***Balanced***
  - Height = O($log_2(n)$) for n nodes
  - Useful for searches



Degenerate binary tree



Balanced binary tree

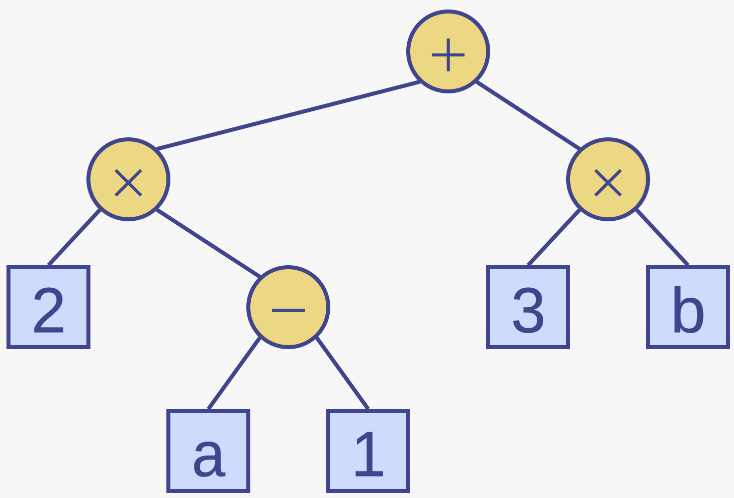
Simple example applications

- arithmetic expressions
- decision processes
- Searching (later lecture)

Arithmetic Expression Tree

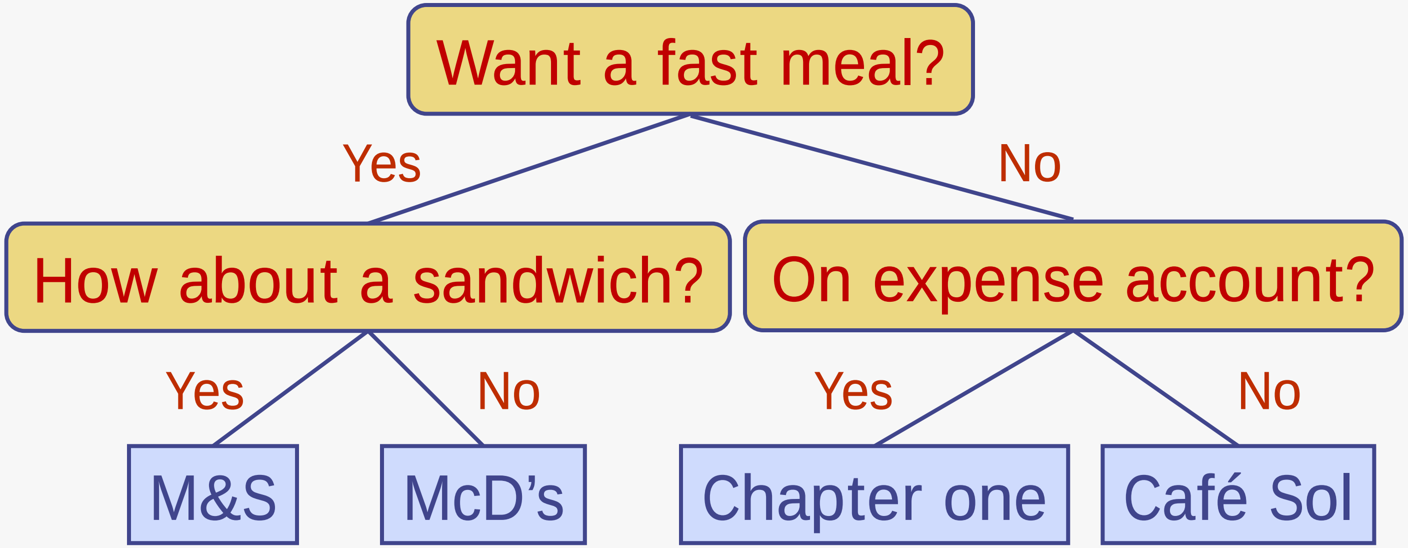
- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands
- Example: arithmetic expression tree for the expression

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



Decision Binary Tree

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



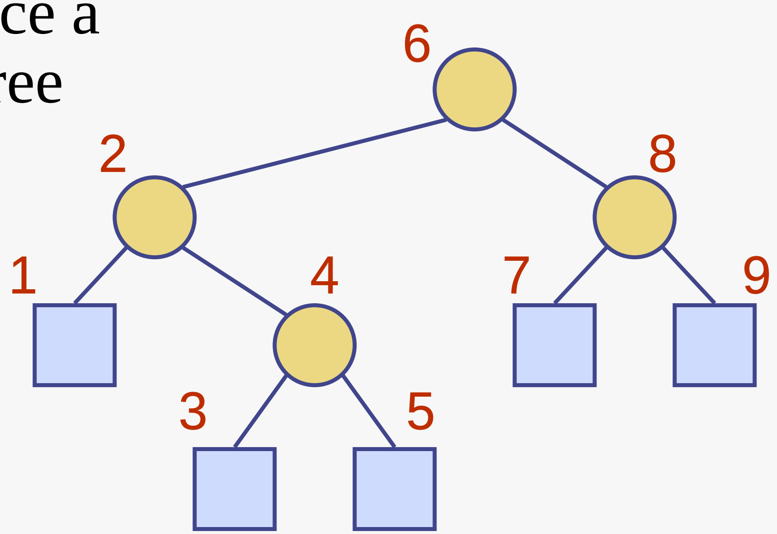
Binary Tree ADT

- Some necessary methods that would be part of the Binary Tree Class:
  - TreeNode **`getLeft()`**
  - TreeNode **`getRight()`**
  - boolean **`hasLeft(p)`**
  - boolean **`hasRight(p)`**
- Update methods may be defined by data structures implementing the Binary Tree Class

Inorder Traversal

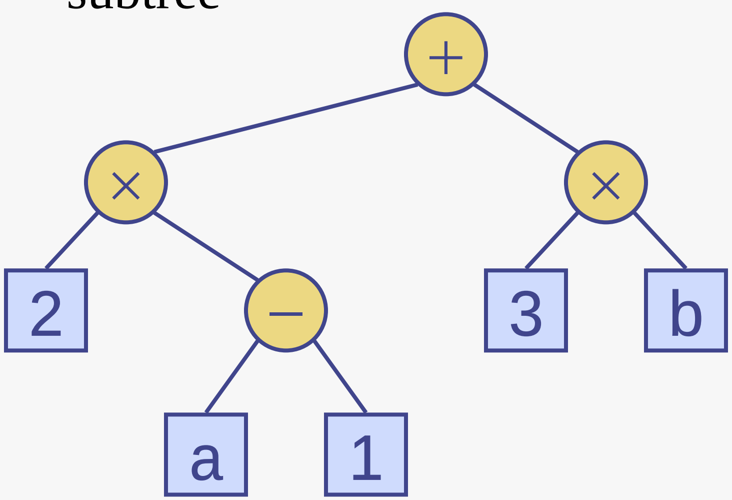
- In an inorder traversal a node is visited after its left subtree and before its right subtree
- Visit nodes in this order
  - Left subtree
  - Root node
  - Right subtree
- Application: to produce a drawing of a binary tree

Algorithm *inOrder*(*v*)
  if *hasLeft* (*v*)
    inOrder (*getLeft* (*v*))
  visit(*v*)
  if *hasRight* (*v*)
    inOrder (*getRight* (*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 *printExpr (v)*

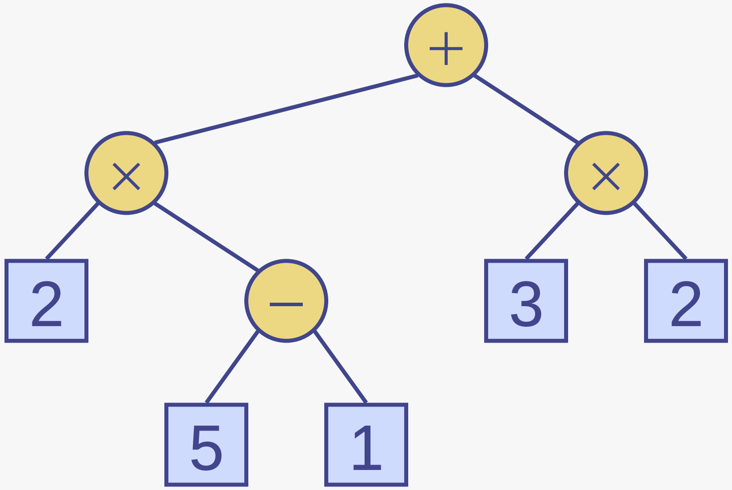
```
if hasLeft (v)
    print(“(’’)
    printExpr(left(v))
print(v.element ())
if hasRight (v)
    printExpr(right(v))
    print (“)’’)
```

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

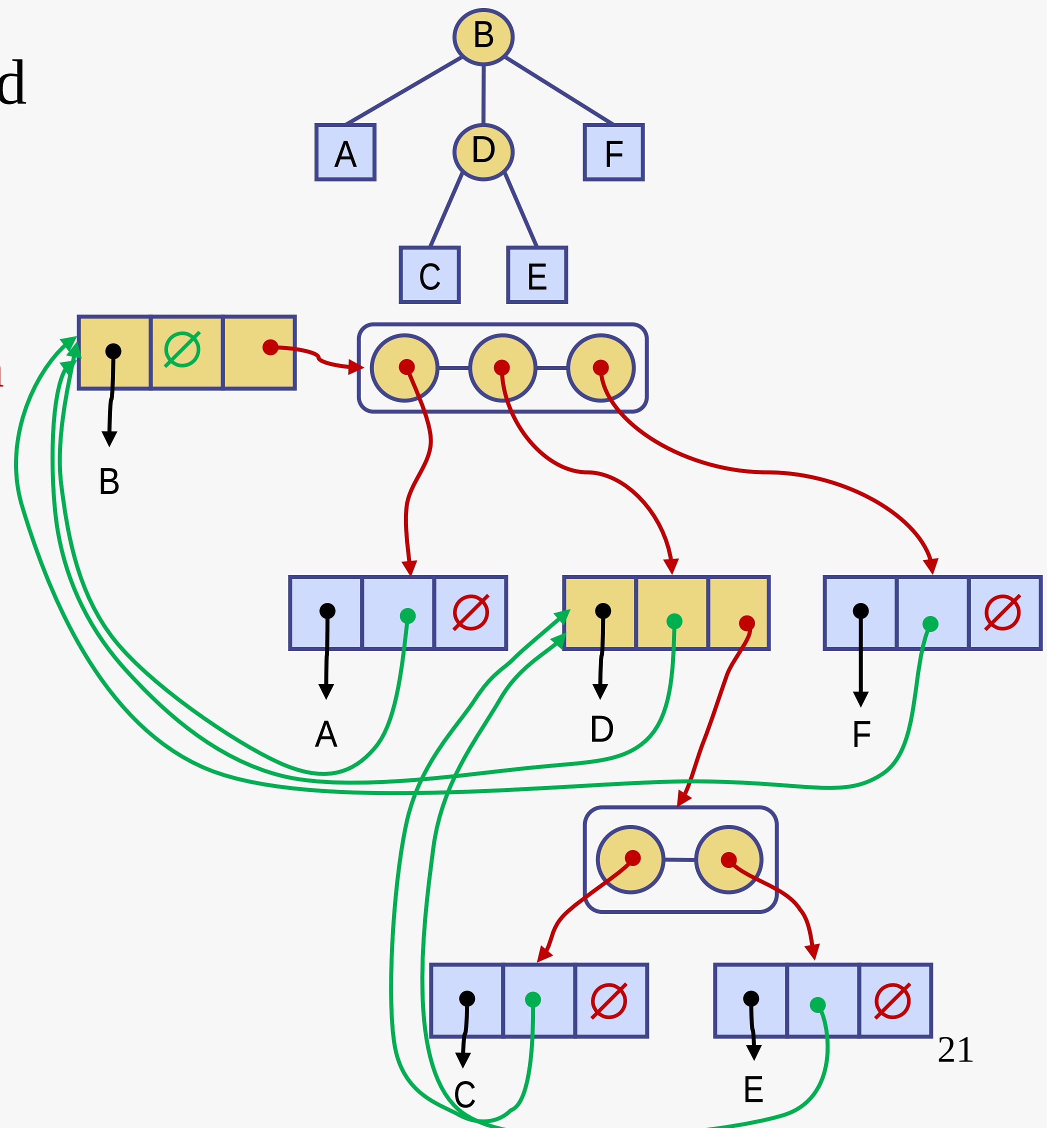
Algorithm *evalExpr*(*v*)
  if *!hasLeft(v) && !hasRight(v)*
    return *v.element* ()
  else
    x ← *evalExpr*(*left* (*v*))
    y ← *evalExpr*(*right* (*v*))
    ◊ ← operator stored at ***v***
    return *x* ◊ *y*



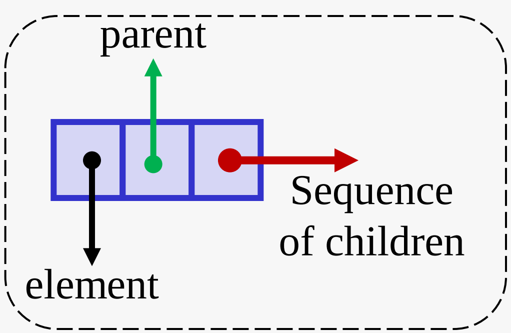
x ◊ *y* ← 5-1
x ◊ *y* ← 2 x (5-1)
x ◊ *y* ← 3 x 2
x ◊ *y* ← 2 x (5-1) + (3 x 2)

Linked Structure for Trees

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

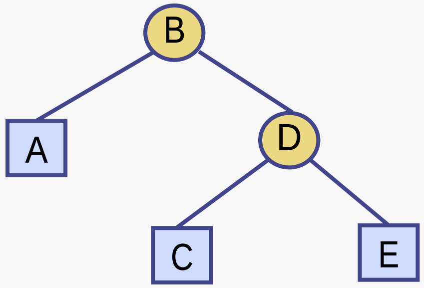


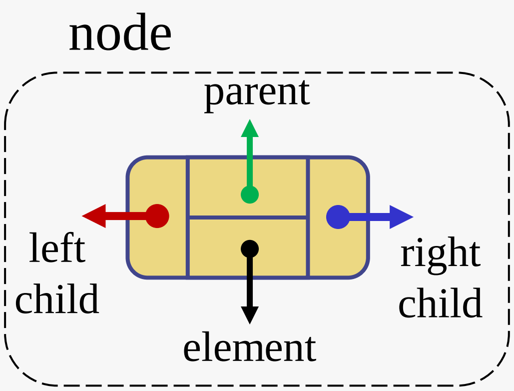
node

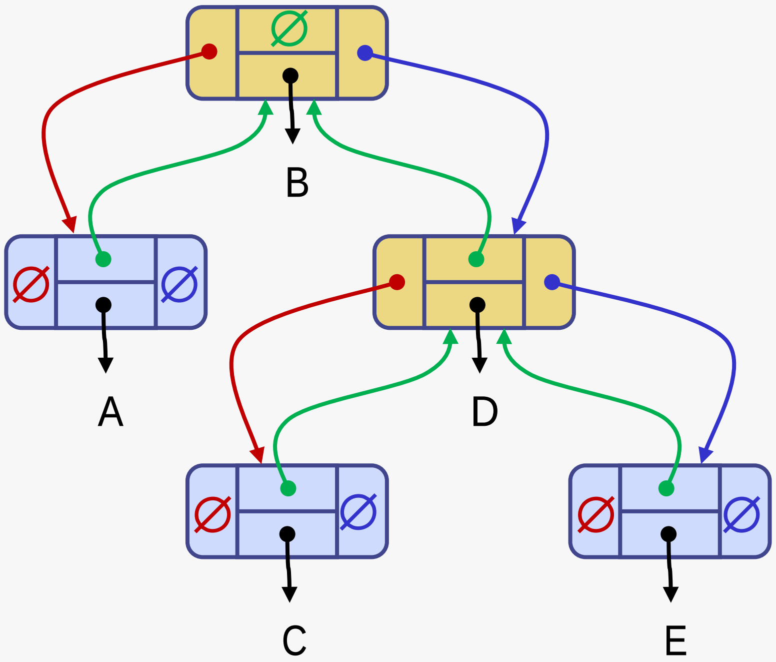


Linked Structure for Binary Trees

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







Acknowledgements

- Some of this material has been taken from a variety of sources
- the most notable of which is from

  Goodrich and Tamassia "Data Structures and Algorithms in Java" John Wiley & Sons.