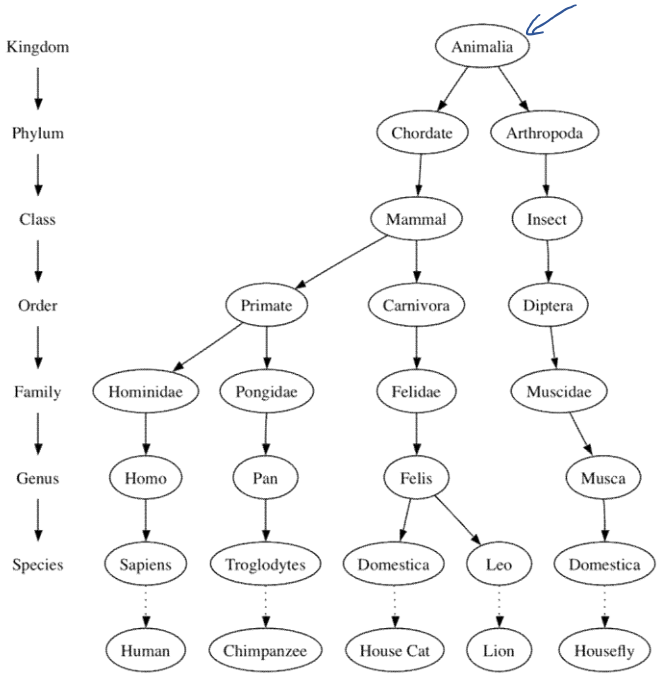
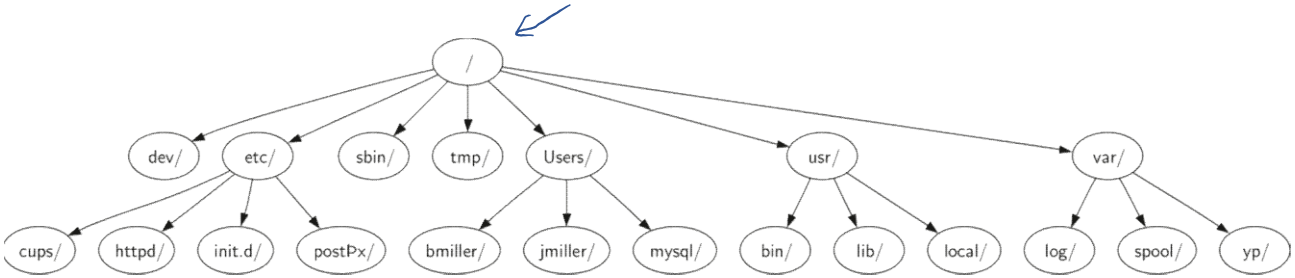
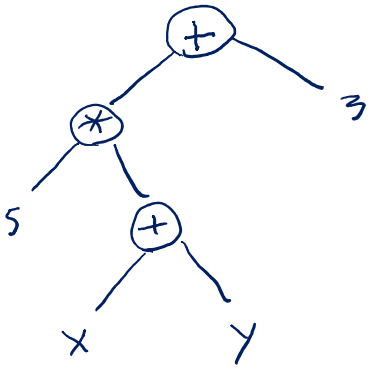
Objectives

- Be able to identify all the following parts of a tree: Node, Edge, Root, Child, Descendant, Path, Parent, Sibling, Subtree, Leaf Node, Level, and Height
- Be able to code the nodes and references representation of a binary tree
- Be able to list and describe the member functions of class BinaryTree
- Be able to code the member functions of class BinaryTree
- Be able to build an expression tree
- Be able to write the code to evaluate an expression tree

Chapter 6: Trees and Tree Algorithms





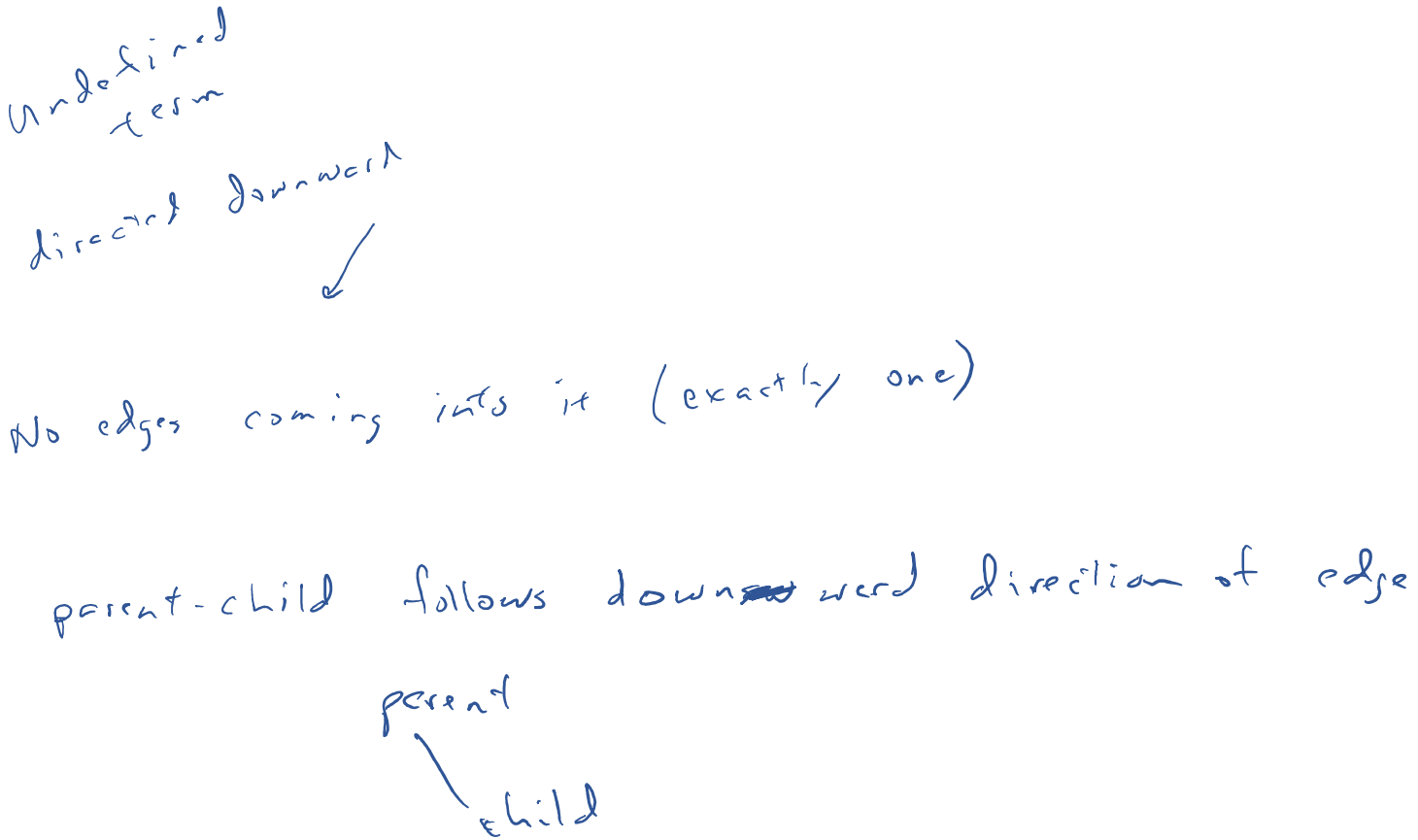


expression tree

recursion tree

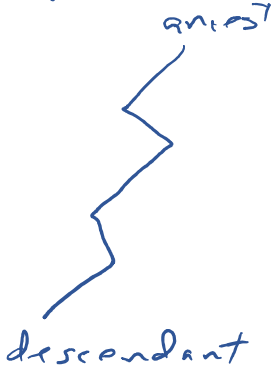
Terms

- Node
- Edge
- Root
- Child



Terms Continued

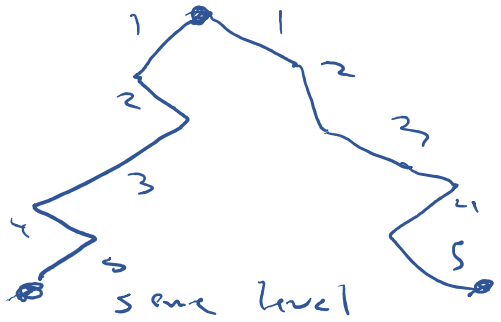
- Descendant A series of parent-child relationships chained together



- Path A series of nodes + edges beg'g with an ancestor + ending with a descendant
- Parent (see child)
- Sibling Two nodes are siblings if they have the same parent.

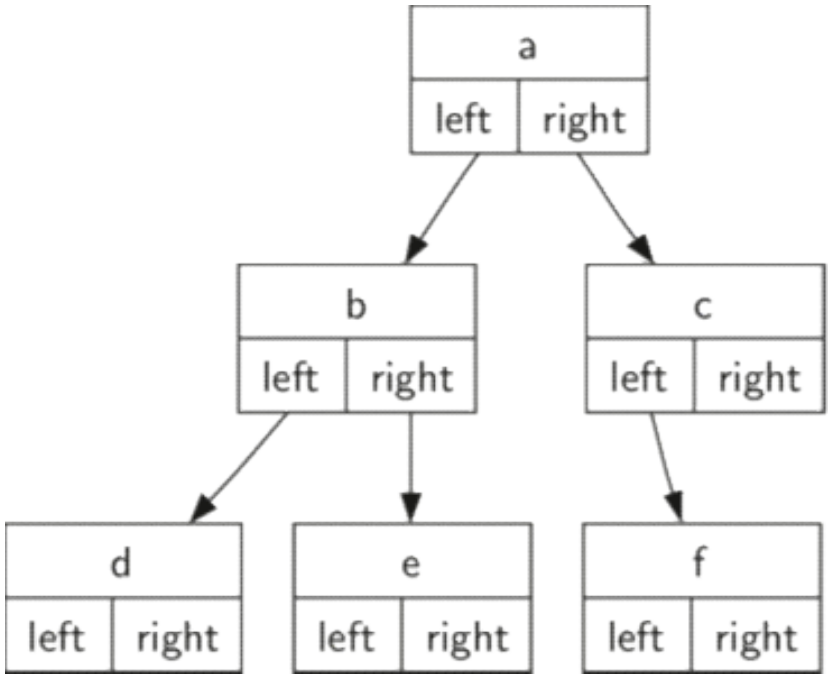
Terms Continued

- Subtree A node and its descendants
- Leaf Node A node w/ no descendants (level 0)
- Level Root is at top level, and if two nodes are at the same level then the path from the root has the same # of edges
- Height Largest level of any node.



A Simple List Representation

Nodes and References

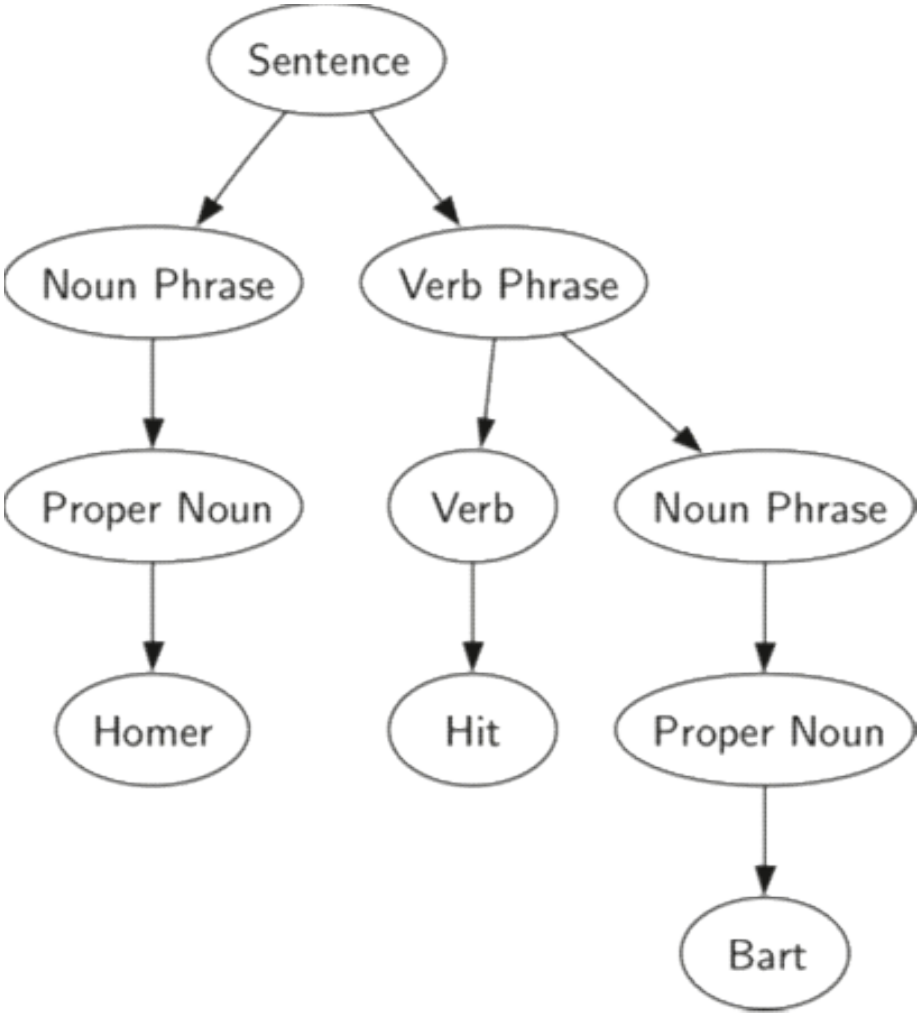


Class BinaryTree

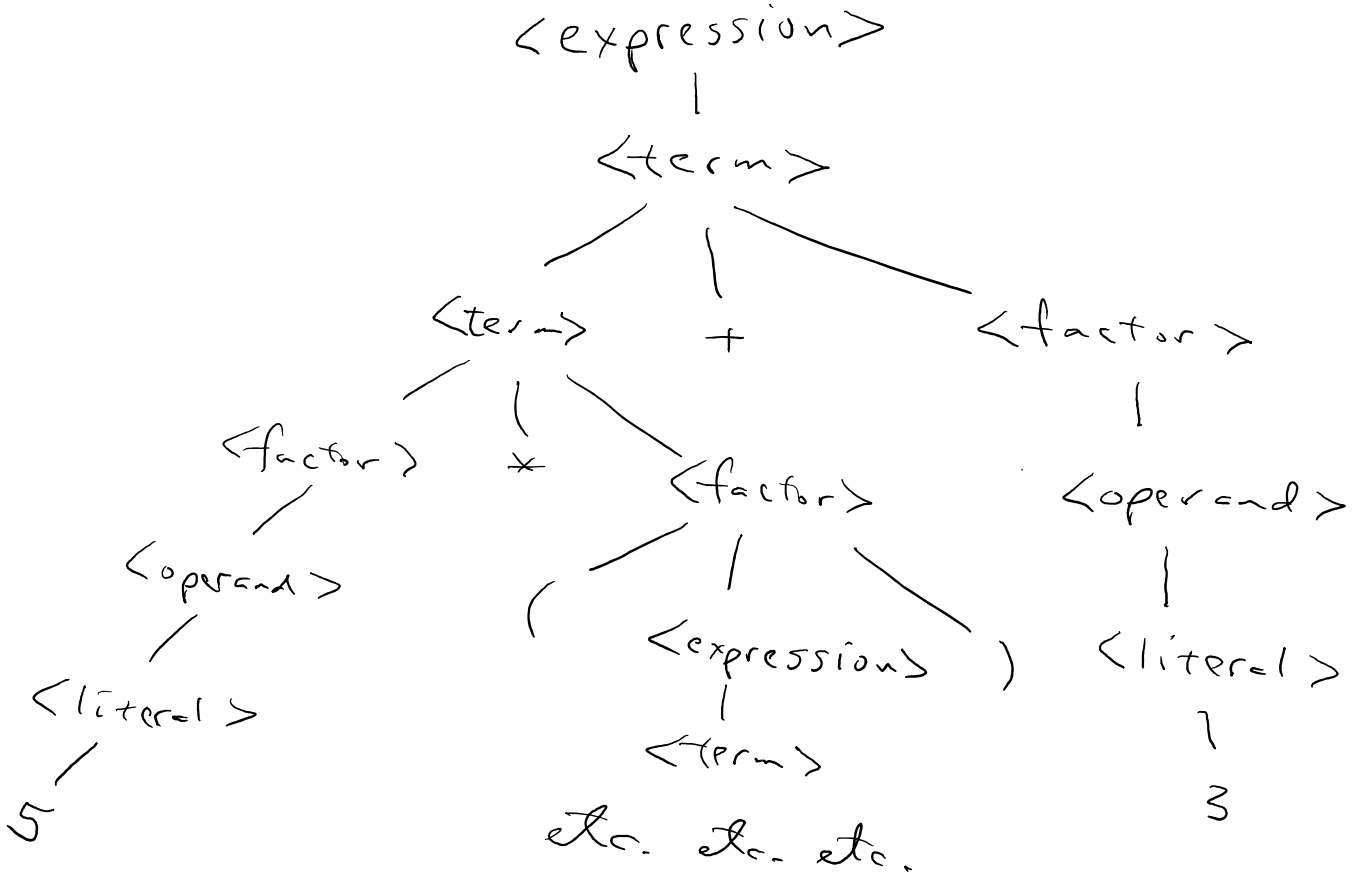
```
class BinaryTree:
    def __init__(self,rootObj):
        self.key = rootObj
        self.leftChild = None
        self.rightChild = None
```

More BinaryTree Member Functions

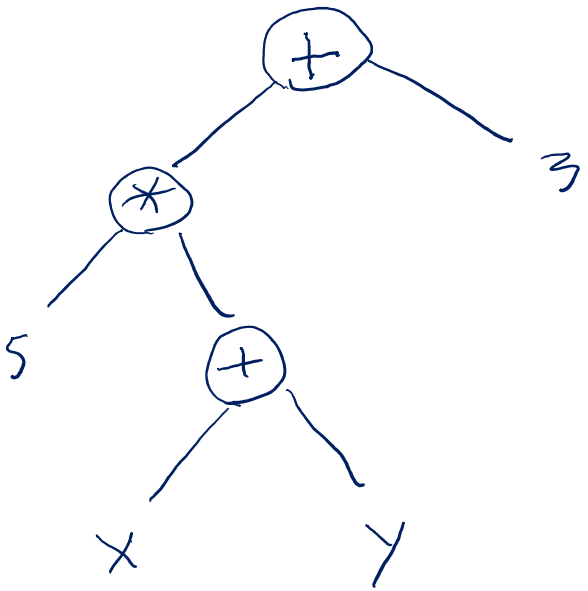
Parse Tree


Sentence
Noun Phrase
Verb Phrase
Proper Noun
Verb
Noun Phrase
Homer
Hit
Proper Noun
Bart

Another Parse Tree



Expression Tree



Conceptual versus explicitly constructed trees

- Parse Trees – Conceptual trees that aid in constructing parsers – never explicitly constructed
- Expression Trees – Can be and often are explicitly constructed, as run-time representations of infix expressions

Algorithm buildParseTree Expression

```
def buildExpressionTree(fpexp):
    fplist = fpexp.split()
    pStack = Stack()
    eTree = BinaryTree('')
    pStack.push(eTree)
    currentTree = eTree
    for i in fplist:
        if i == '(':
            currentTree.insertLeft('')
            pStack.push(currentTree)
            currentTree = currentTree.getLeftChild()
        elif i not in ['+', '-', '*', '/', ')']:
            currentTree.setRootVal(int(i))
            parent = pStack.pop()
            currentTree = parent
        elif i in ['+', '-', '*', '/']:
            currentTree.setRootVal(i)
            currentTree.insertRight('')
            pStack.push(currentTree)
            currentTree = currentTree.getRightChild()
        elif i == ')':
            currentTree = pStack.pop()
        else:
            raise ValueError
    return eTree
```

Direct Evaluation of an Expression Tree

Tree Traversals