Recursion

Presentation Subtitle

Brad Miller David Ranum[1]

[1] Department of Computer Science
Luther College

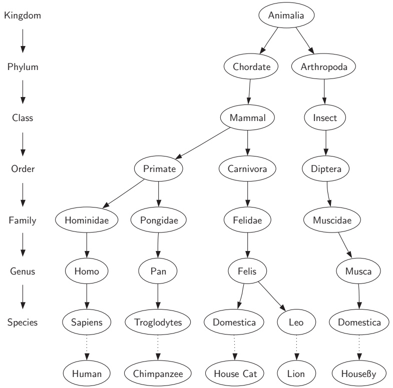
12/19/2005

Outline

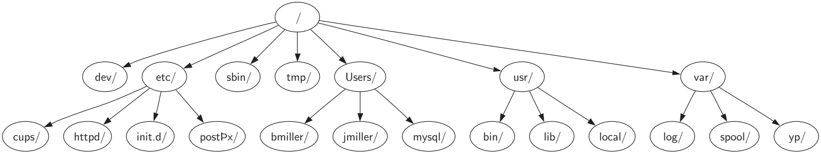
1. Objectives
2. Examples of Trees
3. Vocabulary and Definitions
4. Implementation
   - List of Lists Representation
   - Nodes and References
5. Binary Tree Applications
   - Parse Tree
   - Tree Traversals

- To understand what a tree data structure is and how it is used.
- To see how trees can be used to implement a map data structure.
- To implement trees using a list.
- To implement trees using classes and references.
- To implement trees as a recursive data structure.
- To implement a priority queue using a heap.

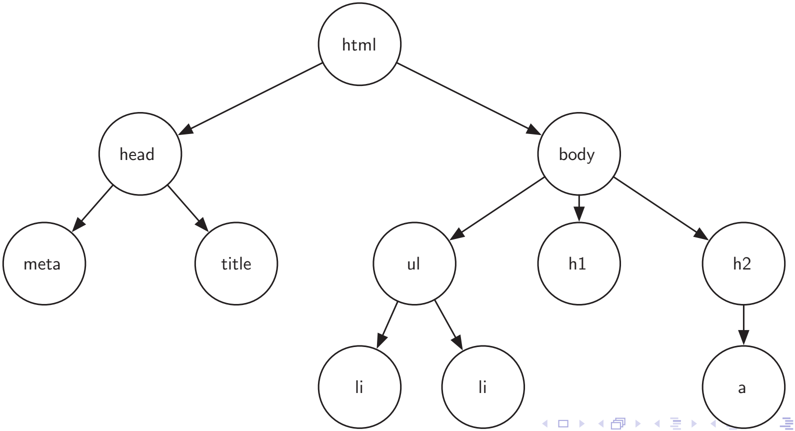
Taxonomy of Some Common Animals Shown as a Tree



A Small Part of the Unix File System Hierarchy

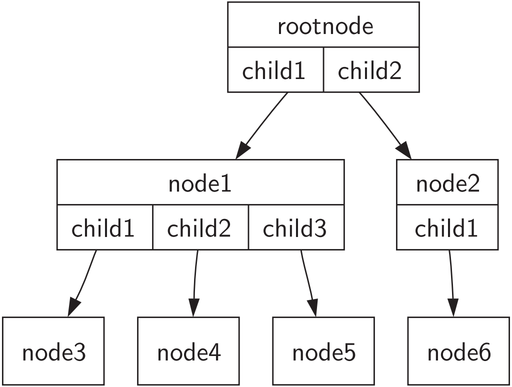


A Tree Corresponding to the Markup Elements of a Webpage

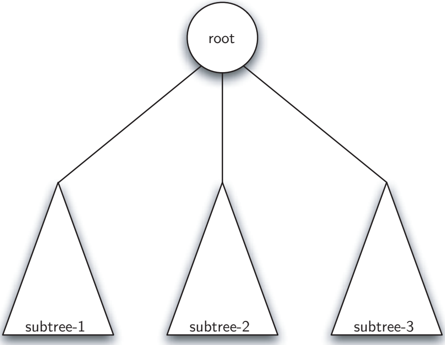


- One node of the tree is designated as the root node.
- Every node *n*, except the root node, is connected by an edge from exactly one other node *p*, where *p* is the parent of *n*.
- A unique path traverses from the root to each node.
- If each node in the tree has a maximum of two children, we say that the tree is a **binary tree**.

A Tree Consisting of a Set of Nodes and Edges



A Recursive Definition of a Tree

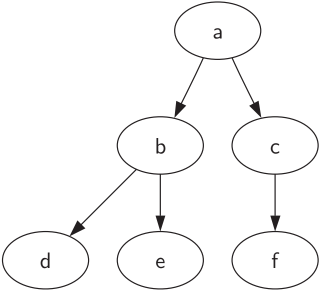


- `BinaryTree()` creates a new instance of a binary tree.
- `getLeftChild()` returns the binary tree corresponding to the left child of the current node.
- `getRightChild()` returns the binary tree corresponding to the right child of the current node.
- `setRootVal(val)` stores the object in parameter `val` in the current node.
- `getRootVal()` returns the object stored in the current node.
- `insertLeft(val)` creates a new binary tree and installs it as the left child of the current node.
- `insertRight(val)` creates a new binary tree and installs it as the right child of the current node.

Outline

1. Objectives
2. Examples of Trees
3. Vocabulary and Definitions
4. Implementation
   - List of Lists Representation
   - Nodes and References
5. Binary Tree Applications
   - Parse Tree
   - Tree Traversals

Representing a Tree As a List of Lists



```
myTree = ['a',   #root
          ['b',  #left subt
           ['d' [], []],
           ['e' [], []] ],
          ['c',  #right sub
           ['f' [], []],
           [] ]
         ]
```

List Functions

```
def BinaryTree(r):
    return [r, [], []]
```

Insert a Left Subtree

```python
def insertLeft(root,newBranch):
    t = root.pop(1)
    if len(t) > 1:
        root.insert(1,[newBranch,t,[]])
    else:
        root.insert(1,[newBranch, [], []])
    return root
```

Insert a Right Subtree

```
def insertRight(root,newBranch):
    t = root.pop(2)
    if len(t) > 1:
        root.insert(2,[newBranch,[],t])
    else:
        root.insert(2,[newBranch,[],[]])
    return root
```

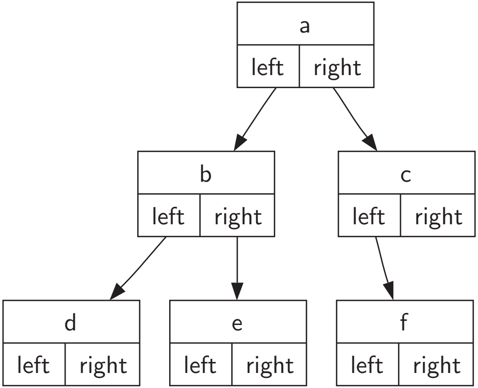
Access Functions for Parts of the Tree

```
def getRootVal(root):
    return root[0]

def setRootVal(root,newVal):
    root[0] = newVal

def getLeftChild(root):
    return root[1]

def getRightChild(root):
    return root[2]
```

Outline

1. Objectives
2. Examples of Trees
3. Vocabulary and Definitions
4. Implementation
   - List of Lists Representation
   - Nodes and References
5. Binary Tree Applications
   - Parse Tree
   - Tree Traversals

A Simple Tree Using a Nodes and References Approach



A Simple Class Definition

```
class BinaryTree:
    def __init__(self,rootObj):
        self.key = rootObj
        self.left = None
        self.right = None
```

Insert a New Left Child

```
    def insertLeft(self,newNode):
        if self.left == None:
            self.left = BinaryTree(newNode)
        else:
            t = BinaryTree(newNode)
            t.left = self.left
            self.left = t
```

Code to Insert a Right Child

```
    def insertRight(self,newNode):
        if self.right == None:
            self.right = BinaryTree(newNode)
        else:
            t = BinaryTree(newNode)
            t.right = self.right
            self.right = t
```

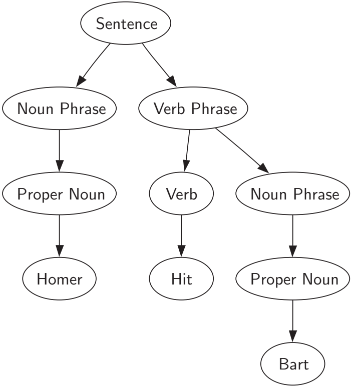
Access Methods for the Binary Tree Class

```
    def getRootVal(self,):
        return self.key

    def setRootVal(self,obj):
        self.key = obj

    def getLeftChild(self):
        return self.left

    def getRightChild(self):
        return self.right
```

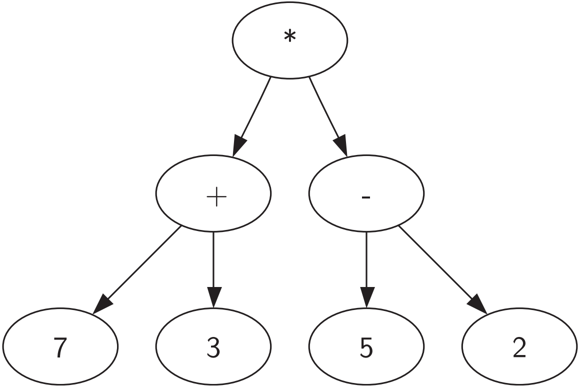
Outline

1. Objectives
2. Examples of Trees
3. Vocabulary and Definitions
4. Implementation
   - List of Lists Representation
   - Nodes and References
5. Binary Tree Applications
   - Parse Tree
   - Tree Traversals

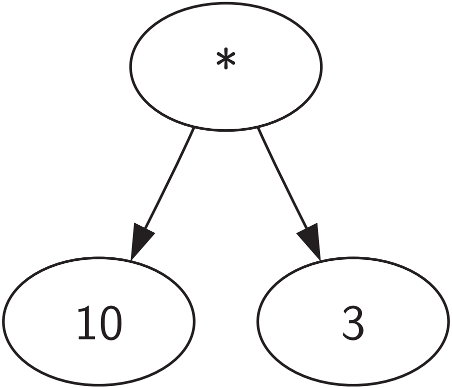
A Parse Tree for a Simple Sentence



Parse Tree for $((7 + 3) * (5 - 2))$



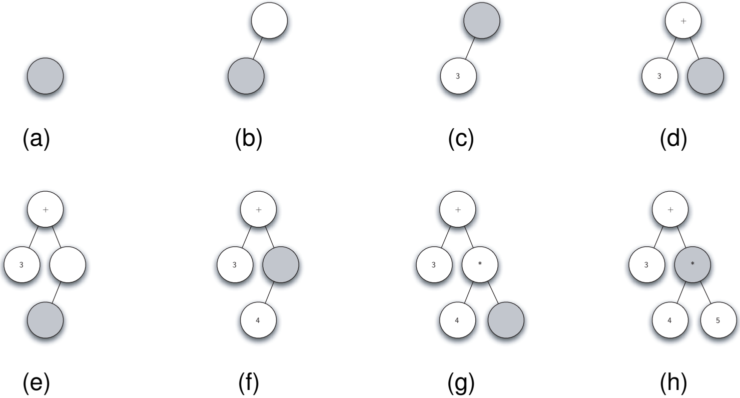
A simplified parse tree for $((7 + 3) * (5 - 2))$



- How to build a parse tree from a fully parenthesized mathematical expression.
- How to evaluate the expression stored in a parse tree.
- How to recover the original mathematical expression from a parse tree.

1. If the current token is a `'('`, add a new node as the left child of the current node, and descend to the left child.
2. If the current token is in the list `['+','-','/','*']`, set the root value of the current node to the operator represented by the current token. Add a new node as the right child of the current node and descend to the right child.
3. If the current token is a number, set the root value of the current node to the number and return to the parent.
4. If the current token is a `')'`, go to the parent of the current node.

Tracing Parse Tree Construction



(a) (b) (c) (d)

(e) (f) (g) (h)

a) Create an empty tree.

b) Read (as the first token. By rule 1, create a new node as the left child of the root. Make the current node this new child.

c) Read 3 as the next token. By rule 3, set the root value of the current node to 3 and go back up the tree to the parent.

d) Read + as the next token. By rule 2, set the root value of the current node to + and add a new node as the right child. The new right child becomes the current node.

e) Read a (as the next token. By rule 1, create a new node as the left child of the current node. The new left child becomes the current node.

f) Read a 4 as the next token. By rule 3, set the value of the current node to 4. Make the parent of 4 the current node.

g) Read * as the next token. By rule 2, set the root value of the current node to * and create a new right child. The new right child becomes the current node

Code to Create a Parse Tree I

```
def buildParseTree(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(eval(i))
            parent = pStack.pop()
            currentTree = parent
```

Code to Create a Parse Tree II

```
        elif i in '+-*/':
            currentTree.setRootVal(i)
            currentTree.insertRight('')
            pStack.push(currentTree)
            currentTree = currentTree.getRightChild()
        elif i == ')':
            currentTree = pStack.pop()
        else:
            print "error:  I don't recognize " + i
    return eTree
```

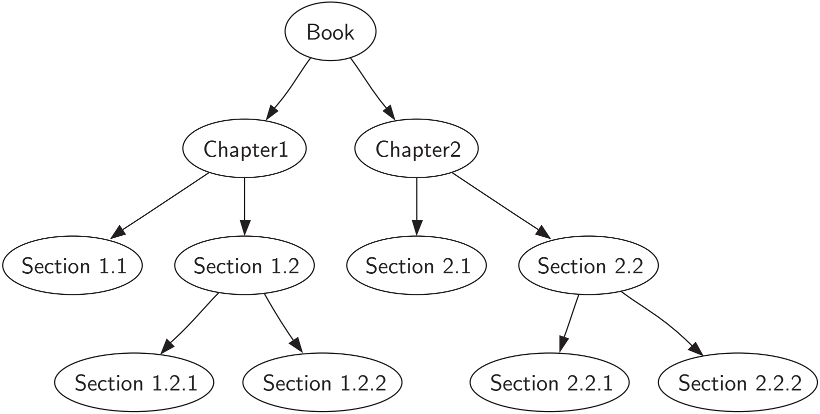
A Recursive Function to Evaluate a Binary Parse Tree

```
def evaluate(parseTree):
    opers = {’+’:operator.add, ’−’:operator.sub,
             ’∗’:operator.mul, ’/’:operator.div}
    leftC = parseTree.getLeftChild()
    rightC = parseTree.getRightChild()

    if leftC and rightC:
        fn = opers[parseTree.getRootVal()]
        return fn(evaluate(leftC),evaluate(rightC))
    else:
        return parseTree.getRootVal()
```

Outline

1. Objectives
2. Examples of Trees
3. Vocabulary and Definitions
4. Implementation
    - List of Lists Representation
    - Nodes and References
5. Binary Tree Applications
    - Parse Tree
    - Tree Traversals

Representing a Book As a Tree



External Function Implementing Preorder Traversal of a Tree I

```
def preorder(tree):
    if tree:
        print tree.getRootVal()
        preorder(tree.getLeftChild())
        preorder(tree.getRightChild())
```

Preorder Traversal Implemented as a Method of `BinaryTree` I

```
def preorder(self):
    print self.key
    if self.left:
        self.left.preorder()
    if self.right:
        self.right.preorder()
```

Postorder Traversal Algorithm I

```
def postorder(tree):
    if tree != None:
        postorder(tree.getLeftChild())
        postorder(tree.getRightChild())
        print tree.getRootVal()
```

Postorder Evaluation Algorithm I

```
def postordereval(tree):
    opers = {'+':operator.add, '-':operator.sub,
             '*':operator.mul, '/':operator.div}
    res1 = None
    res2 = None
    if tree:
        res1 = postordereval(tree.getLeftChild())
        res2 = postordereval(tree.getRightChild())
        if res1 and res2:
            return opers[tree.getRootVal()](res1,res2)
        else:
            return tree.getRootVal()
```

Inorder Traversal Algorithm I

```
def inorder(tree):
    if tree != None:
        inorder(tree.getLeftChild())
        print tree.getRootVal()
        inorder(tree.getRightChild())
```

Modified Inorder Traversal to Print Fully Parenthesized Expression I

```
def printexp(tree):
    sVal = ""
    if tree:
        sVal = ’(’ + printexp(tree.getLeftChild())
        sVal = sVal + str(tree.getRootVal())
        sVal = sVal + printexp(tree.getRightChild())+’)’
    return sVal
```