Tree

What is a tree
什么是树

The example of a tree
树的例子

In this chapter, we discuss a basic "nonlinear" data structure - tree;
本章我们来讨论一种基本的“非线性”数据结构——树；

Trees are widely used in various fields of computer science
树在计算机科学的各个领域中被广泛应用

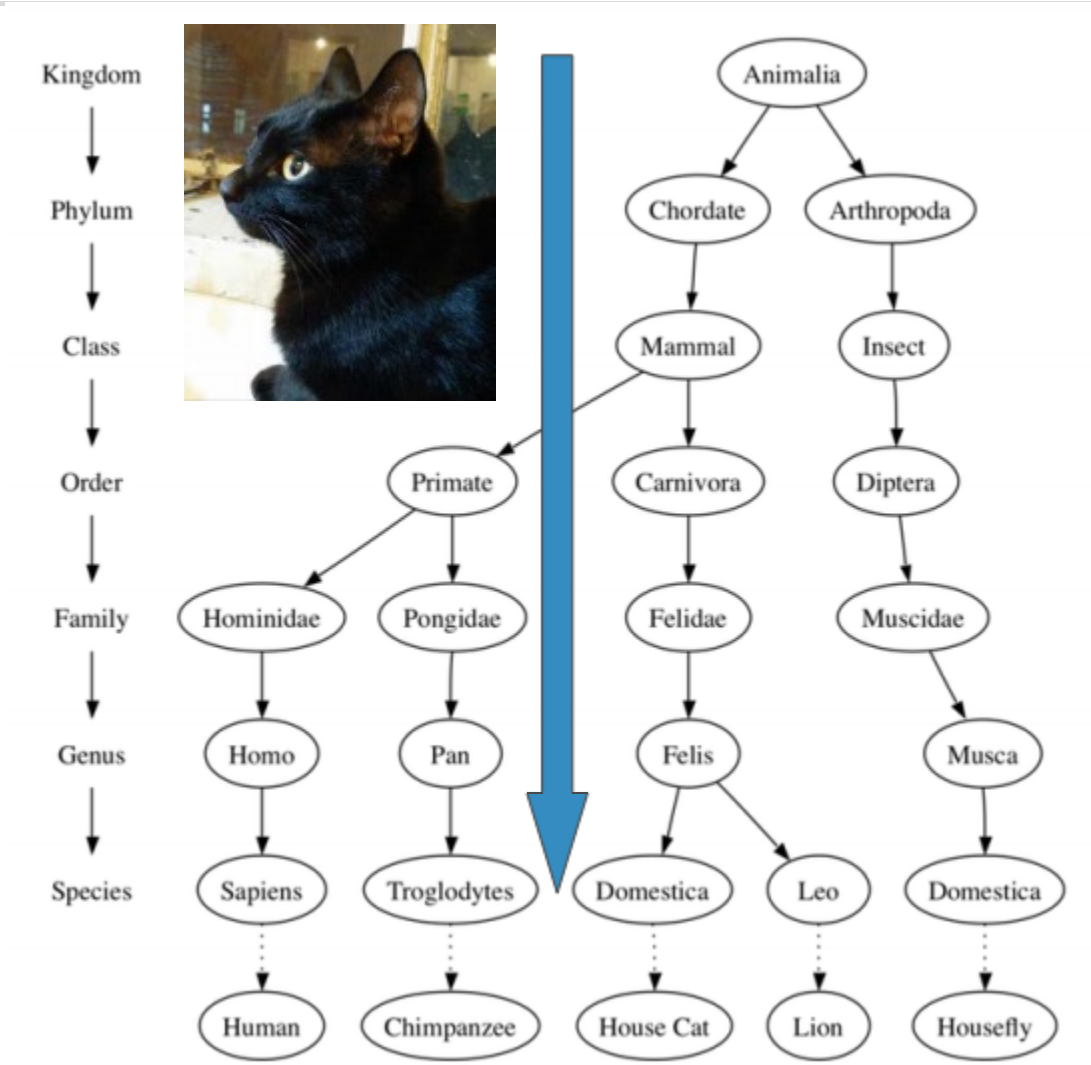
Operating system, graphics, database system, computer network
操作系统、图形学、数据库系统、计算机网络

Like trees in nature, the data structure trees are divided into three parts: roots, branches and leaves
跟自然界中的树一样，数据结构树也分为：根、枝和叶等三个部分

The illustration of the general data structure places the roots above and the leaves below
一般数据结构的图示把根放在上方，叶放在下方

Example of a tree: a biological species classification system

树的例子：生物学物种分类体系



Example of a tree: a biological species classification system
树的例子：生物学物种分类体系

First, the classification system is
首先分类体系是

Hierarchized
层次化的

A tree is a single, hierarchical structure
树是一种分层结构

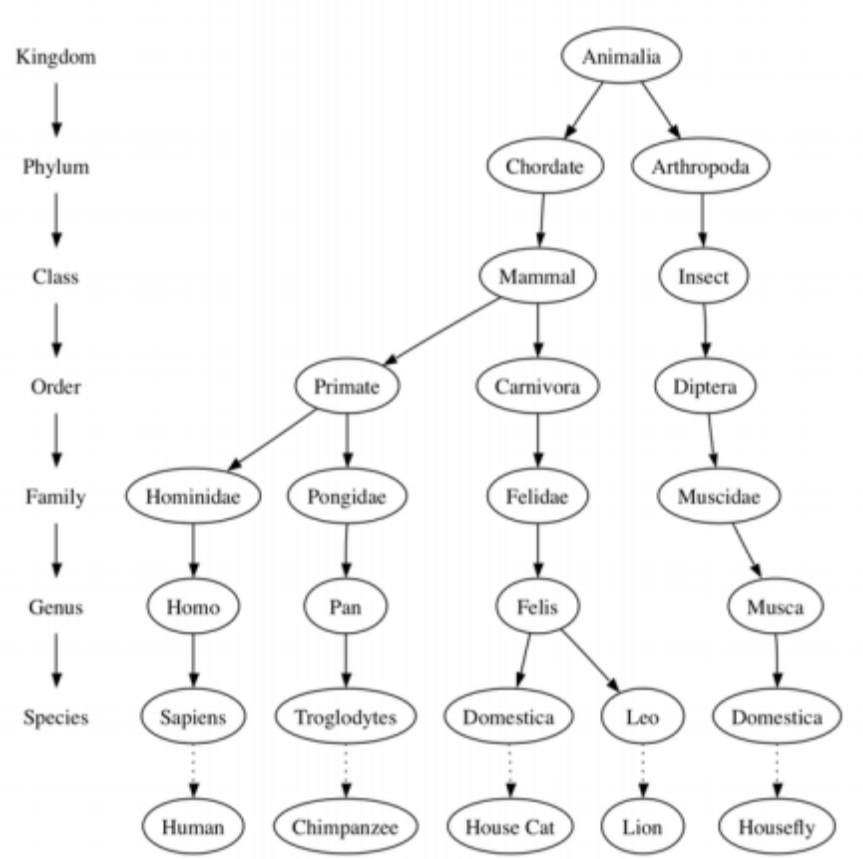
The closer to the top is, the more the layer
越接近顶部的层越

universal is
普遍

The closer the bottom layer is, the more the layer
越接近底部的层越

unique is
独特

Borders, phyla, class, order, family, genus, and species
界、门、纲、目、科、属、种



Example of a tree: a biological species classification system
树的例子：生物学物种分类体系

The second feature of the classification tree: the children of one node are isolated and independent from those of the other

分类树的第二个特征：一个节点的子节点与另一个节点的子节点相互之间是隔离、独立的

Both cat genera Felis and fly Musca have the node of the same name “Domestica”

猫属Felis和蝇属Musca下面都有Domestica的同名节点

But if there is no correlation with each other, and one Domestica can be modified without affecting the other.

但相互之间并无任何关联，可以修改其中一个Domestica而不影响另一个。

Example of a tree: a biological species classification system

树的例子：生物学物种分类体系

The third feature of the classification tree is that each leaf node is unique

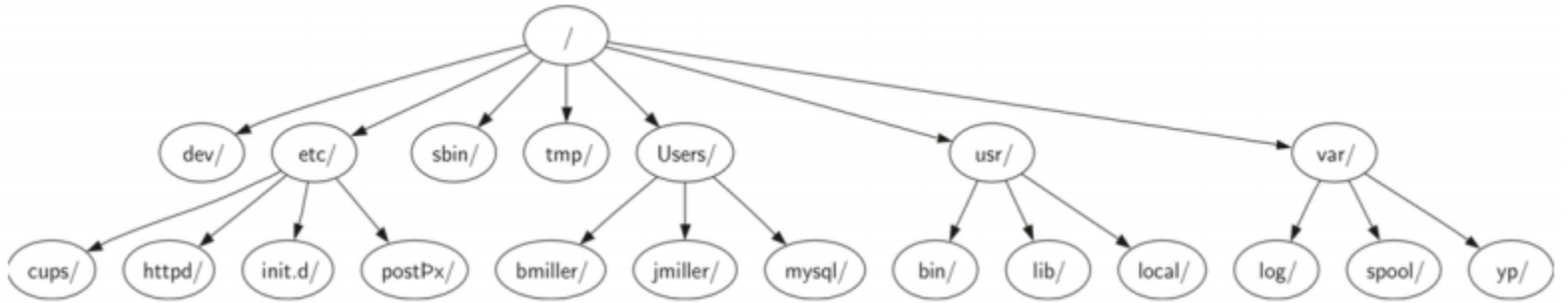
分类树的第三个特征：每一个叶节点都具有唯一性

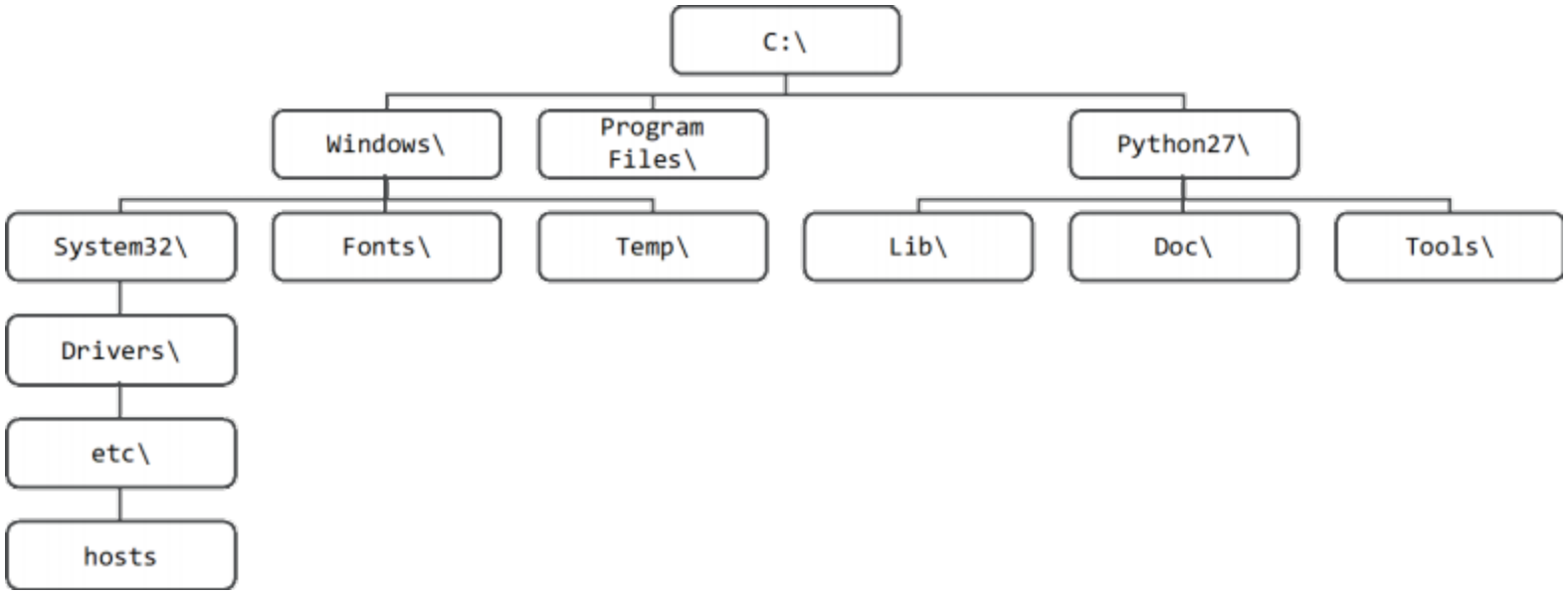
Each species can be uniquely identified by a full path from the root to each species, the animal kingdom-> chordate-> mammalian-> carnivorous-> cat-> cat-> domestic cat species

可以用从根开始到达每个种的完全路径来唯一标识每个物种，动物界->脊索门->哺乳纲->食肉目->猫科->猫属->家猫种

Animalia->Chordate->Mammal->Carnivora->Felidae->Felis->Domestica

Example of a tree: File system
树的例子：文件系统

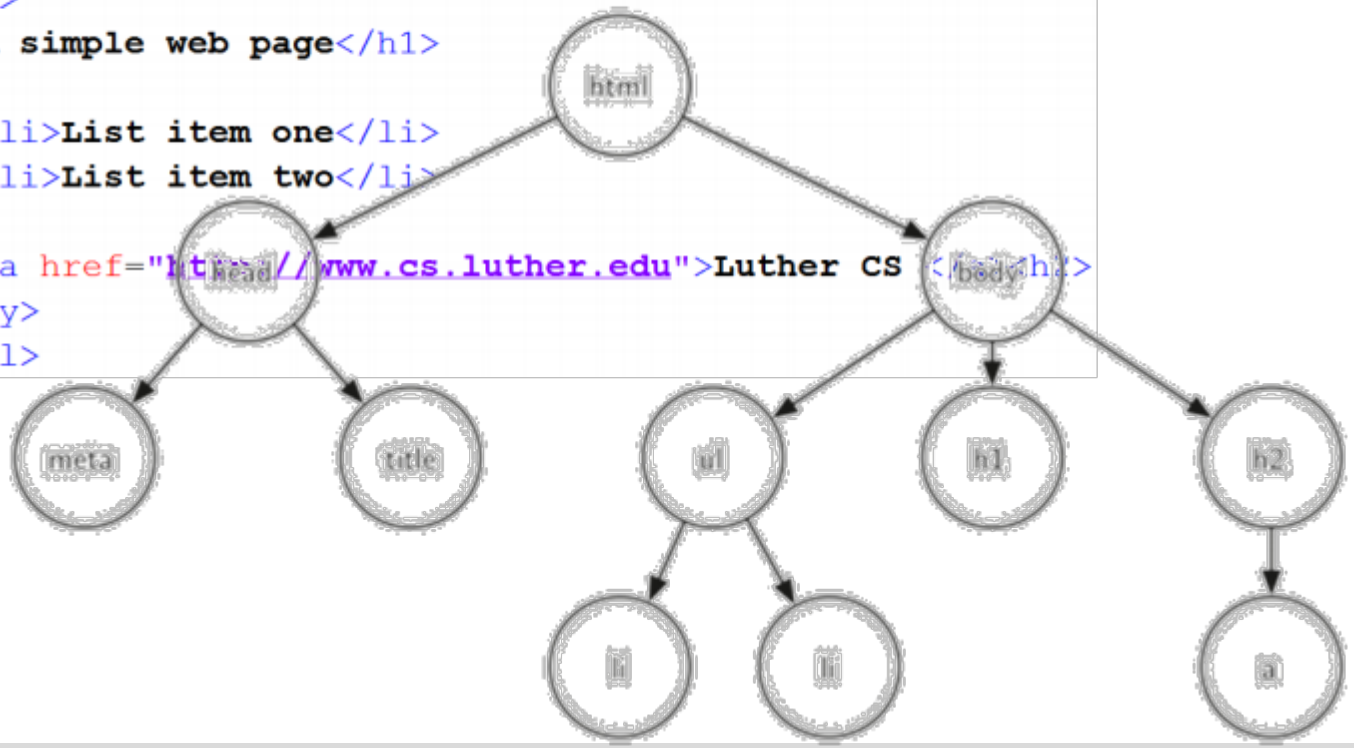




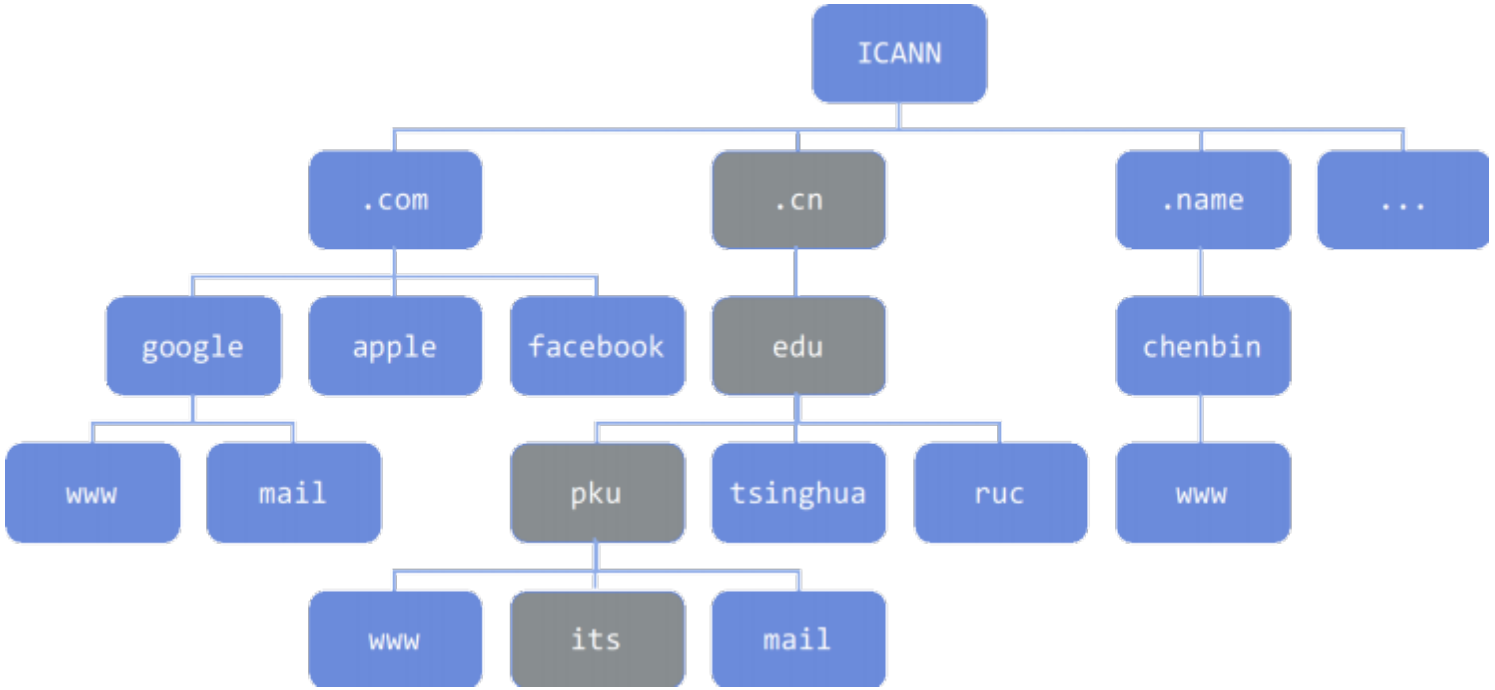
Example of a tree: an HTML document (nested tag)

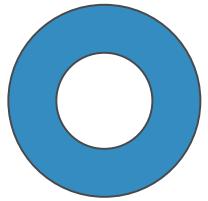
树的例子：HTML文档(嵌套标记)

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type"
          content="text/html; charset=utf-8" />
    <title>simple</title>
</head>
<body>
<h1>A simple web page</h1>
<ul>
    <li>List item one</li>
    <li>List item two</li>
</ul>
<h2><a href="http://www.cs.luther.edu">Luther CS</a></h2>
</body>
</html>
```



Example of a tree: Domain name system
树的例子：域名体系





Tree-structure-related terms
树结构相关术语

Tree-structure-related terms
树结构相关术语

Node: constitutes the basic part of the tree
节点Node：组成树的基本部分

Each node has a name, or a "key value," and the node can also save additional data items, which vary according to the different application

每个节点具有名称，或“键值”，节点还可以保存额外数据项，数据项根据不同的应用而变

Edge: Edge is another basic part of the tree
边Edge：边是组成树的另一个基本部分

Each edge just connects the two nodes, indicating that the nodes have the correlation between them, and the edges have the direction of entry and exit;

每条边恰好连接两个节点，表示节点之间具有关联，边具有出入方向；

Each node (excluding the root node) has an incoming edge from another node;

每个节点(除根节点)恰有一条来自另一节点的入边；

Each node can have multiple outgoing edges connected to other nodes.

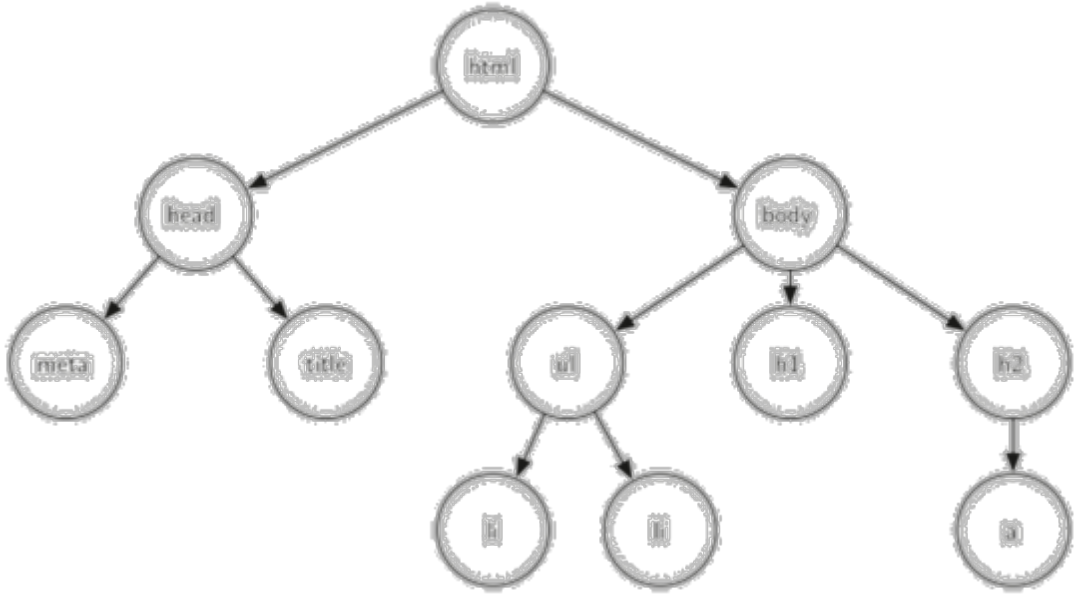
每个节点可以有多条连到其它节点的出边。

Tree-structure-related terms
树结构相关术语

Root: The only node in the tree that has no incoming edges
根Root：树中唯一一个没有入边的节点

Path: A sequential table of nodes connected by edges
路径Path：由边依次连接在一起的节点的有序列表

For example: HTML-> BODY-> UL-> L I, is a path
如：HTML->BODY->UL->LI，是一条路径



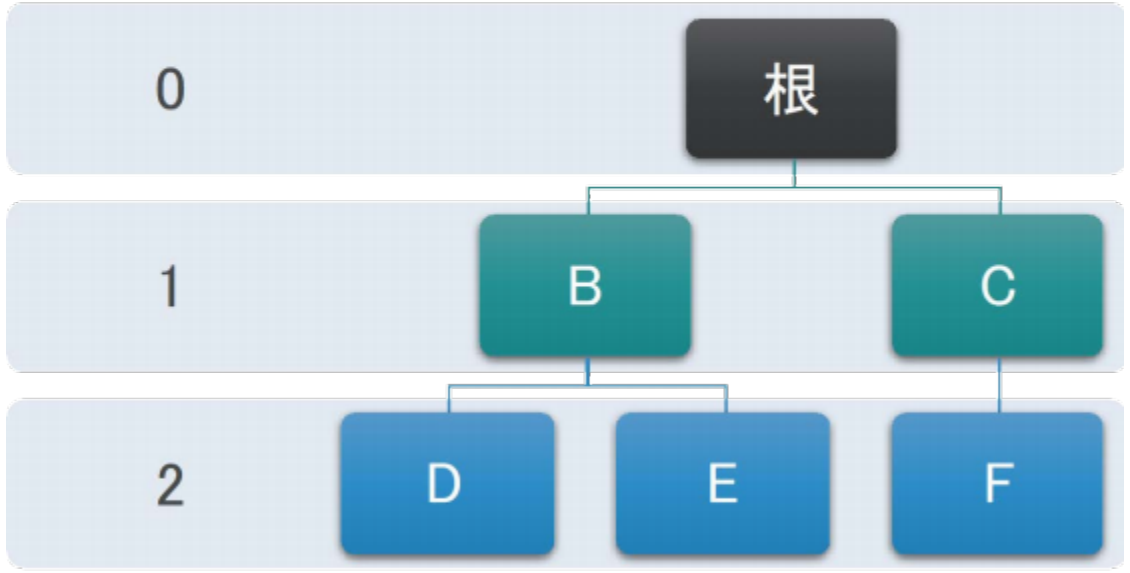
Tree-structure-related terms
树结构相关术语

Child node : The incoming edges all come from several nodes of the same node, which is called the child node of this node

子节点：入边均来自于同一个节点的若干节点，称为这个节点的子节点

Parent node: A node is the parent node where all its outgoing edges are connected

父节点Parent：一个节点是其所有出边所连接节点的父节点



Tree-structure-related terms

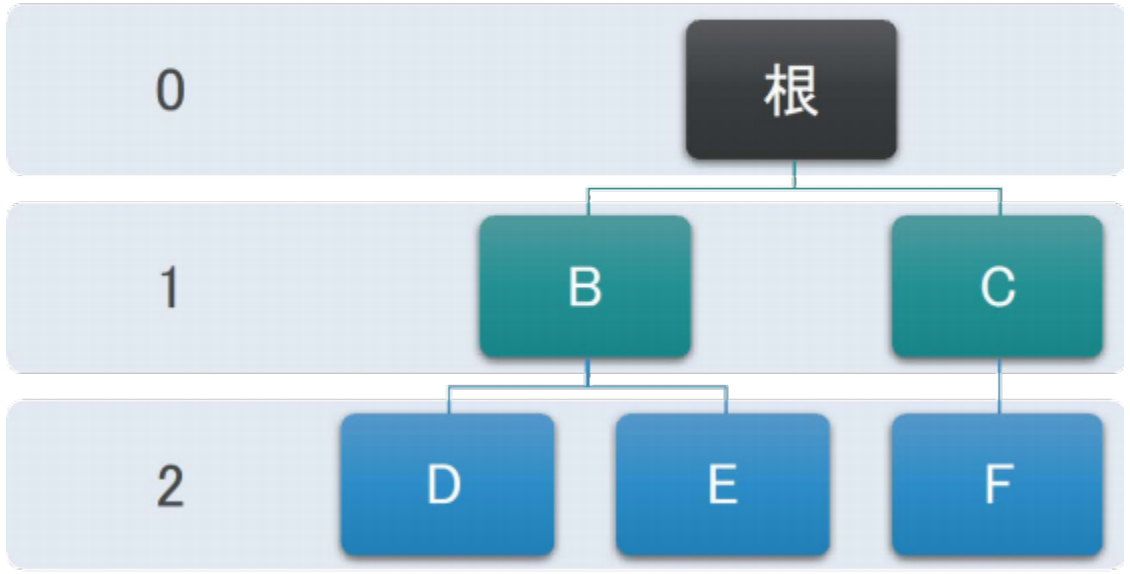
树结构相关术语

Sibling node: The nodes with the same parent node are called sibling nodes

兄弟节点：具有同一个父节点的节点之间称为兄弟节点

Subtree: a collection of a node, all its descendant nodes, and the associated edges

子树Subtree：一个节点和其所有子孙节点，以及相关边的集合



Tree-structure-related terms
树结构相关术语

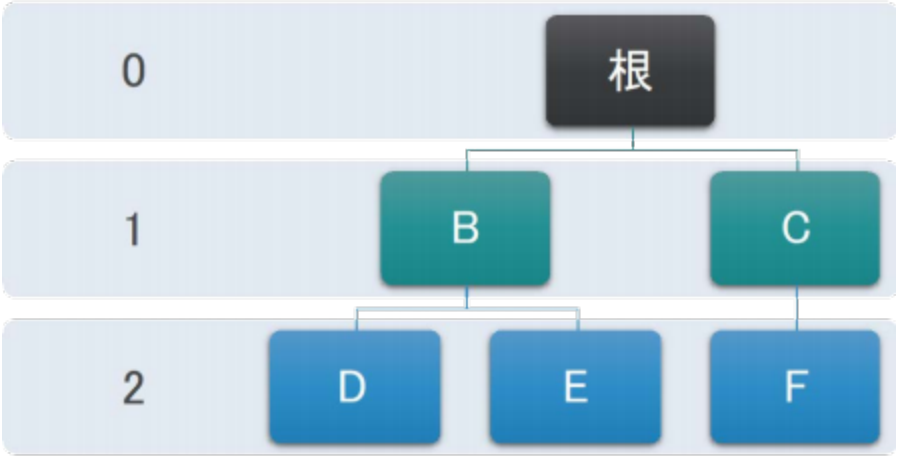
Leaf node: A node without a child node is called a leaf node
叶节点：没有子节点的节点称为叶节点

Level: The path to a node starting from the root node, and the number of edges included, is called the level of the node.
层级：从根节点开始到达一个节点的路径，所包含的边的数量，称为这个节点的层级。

for example, D's level is 2 and root-node's level is 0
如D的层级为2，根节点的层级为0

Height: The maximum level of all the nodes in a tree is called the height of the tree
高度：树中所有节点的最大层级称为树的高度

The height of the tree on the right figure is 2
如右图树的高度为2



Definition of the tree 1
树的定义1

A tree consists of a number of nodes and edges that connect the nodes in pairs, and has the following attributes:

树由若干节点，以及两两连接节点的边组成，并有如下性质

one of the nodes is set as the root;

其中一个节点被设定为根；

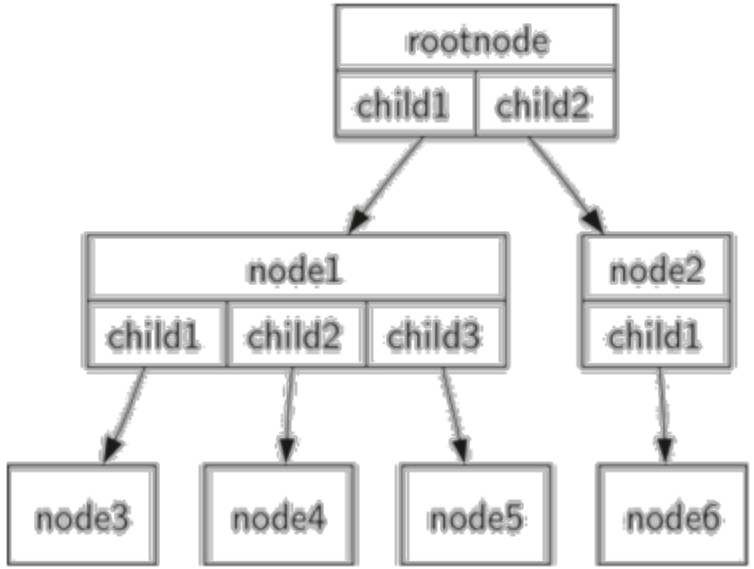
Each node n (except the root node) is just connected to an edge from the node p, which is the parent node of n;

每个节点n（除根节点），都恰连接一条来自节点p的边，p是n的父节点；

The path of each node starting from the root is unique.If each node has at most two children, such a tree is called a "binary tree".

每个节点从根开始的路径是唯一的。

如果每个节点最多有两个子节点，这样的树称为“二叉树”

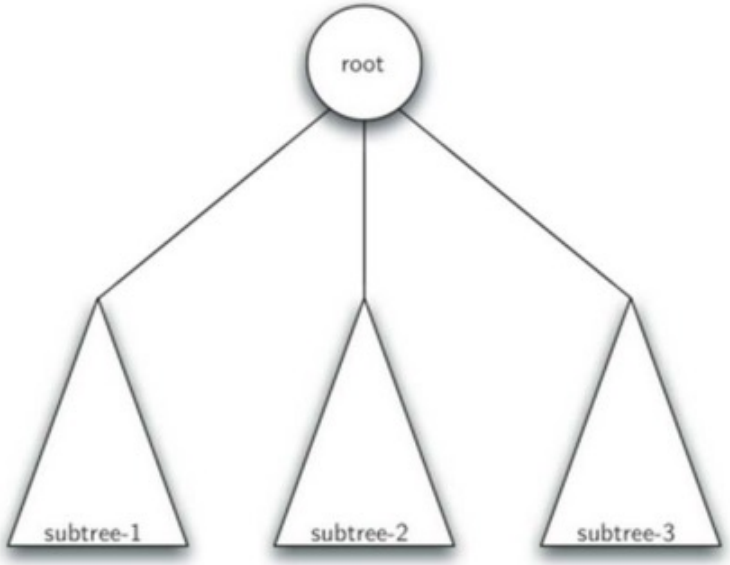


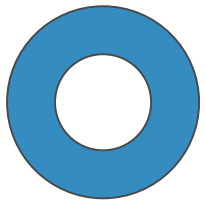
Definition 2 (recursive definition)
树的定义2(递归定义)

The tree is:
树是：

null set;
空集；

Or consists of a root node and 0 or more subtrees (where the subtree is also a tree), each subtree is connected by an edge from the root-to-root node
或者由根节点及0或多个子树构成(其中子树也是树)，每个子树的根到根节点具有边相连。





Nested list implementation of the tree
树的嵌套列表实现

Implement the tree: the nested list method
实现树：嵌套列表法

First, we try to use Python List to realize the binary tree tree data structure;

首先我们尝试用PythonList来实现二叉树树数据结构；

Nested list to implement binary tree recursively, consisting of list with three elements:

递归的嵌套列表实现二叉树，由具有3个元素的列表实现：

- The first element is the value of the root node;
  第1个元素为根节点的值；
- The second element is the left subtree (so also a list);
  第2个元素是左子树(所以也是一个列表)；
- The third element is the right subtree (so also a list).
  第3个元素是右子树(所以也是一个列表)。

```
[root, left, right]
```

Implement the tree: the nested list method
实现树：嵌套列表法

In the example on the right, a 6-node binary tree
以右图的示例，一个6节点的二叉树

The roots are myTree [0], the left subtree myTree [1], and the right subtree myTree [2]

根是myTree[0]，左子树myTree[1]，右子树myTree[2]

Advantages of the nested list method
嵌套列表法的优点

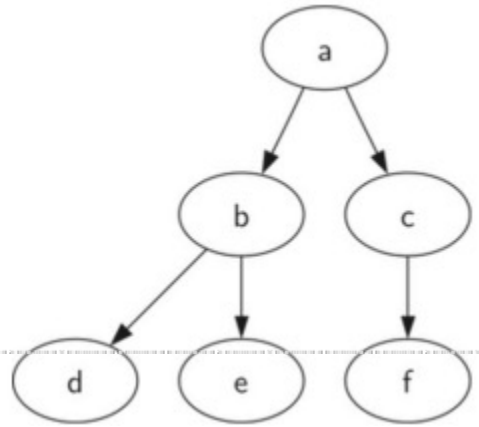
The subtree has the same structure as the tree, and is a recursive data structure

子树的结构与树相同，是一种递归数据结构

It is easy to extend to a multifork tree, just by adding list elements

很容易扩展到多叉树，仅需要增加列表元素即可

```
1  myTree = ['a', # 树根
2            ['b', # 左子树
3             ['d', [], []],
4             ['e', [], []] ],
5            ['c', # 右子树
6             ['f', [], []],
7             [] ]
8            ]
```



Implement the tree: the nested list method
实现树：嵌套列表法

We assisted with nested lists by defining a series of functions
我们通过定义一系列函数来辅助操作嵌套列表

- The BinaryTree creates a binary tree with only a root node. The insertLeft / insertRight inserts the new node into the tree as its direct left / right child node
  `BinaryTree`创建仅有根节点的二叉树`insertLeft/insertRight`将新节点插入树中作为其直接的左/右子节点
- The get/setRootVal takes or returns the root node
  `get/setRootVal`则取得或返回根节点
- The getLeft/RightChild returns to the left/right subtree
  `getLeft/RightChild`返回左/右子树

Nested List method code
嵌套列表法代码

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

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

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

Nested List method code
嵌套列表法代码

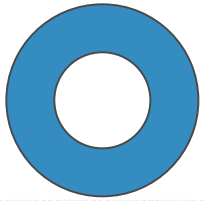
```
def getRootVal(root):
    return root[0]

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

def getLeftChild(root):
    return root[1]

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

Implement the tree: the nested list method

实现树：嵌套列表法

```
r = BinaryTree(3)
insertLeft(r,4)
insertLeft(r,5)
insertRight(r,6)
insertRight(r,7)
l = getLeftChild(r)
print(l)

setRootVal(l,9)
print(r)
insertLeft(l,11)
print(r)
print(getRightChild(getRightChild(r)))
```

```
>>>
[5, [4, [], []], []]
[3, [9, [4, [], []], []], [7, [], [6, [], []]]]
[3, [9, [11, [4, [], []], []], []], [7, [], [6, [], []]]]
[6, [], []]
>>>
```



A linked-list implementation of the tree
树的链表实现

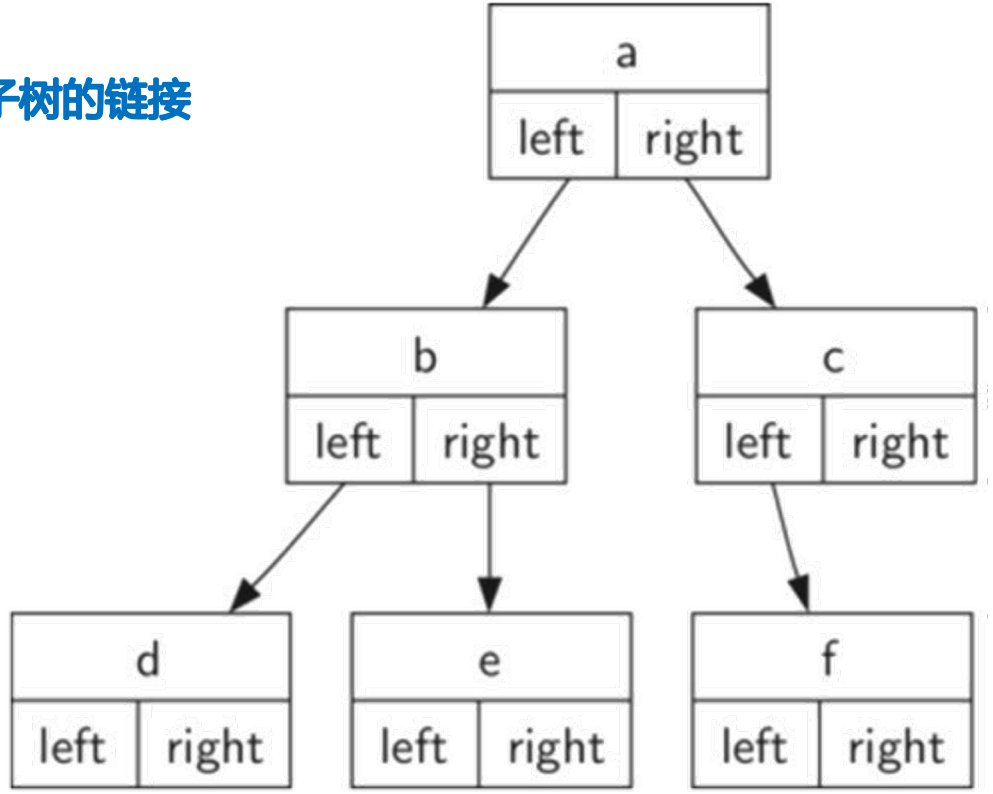
Implement the tree: the node-linking method
实现树：节点链接法

You can also implement the tree by using the node-linking method

同样可以用节点链接法来实现树

Each node holds the data items of the root node, and the links to the left and right subtrees

每个节点保存根节点的数据项，以及指向左右子树的链接



Implement the tree: the node-linking method
实现树：节点链接法

Define a BinaryTree class
定义一个BinaryTree类

Member key holds the root node data item

成员key保存根节点数据项

Member left / rightChild saves the reference to the left / right subtree (also the BinaryTree object)

成员left/rightChild则保存指向左/右子树的引用(同样是BinaryTree对象)

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

Implement the tree: the node-linking method
实现树：节点链接法

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

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

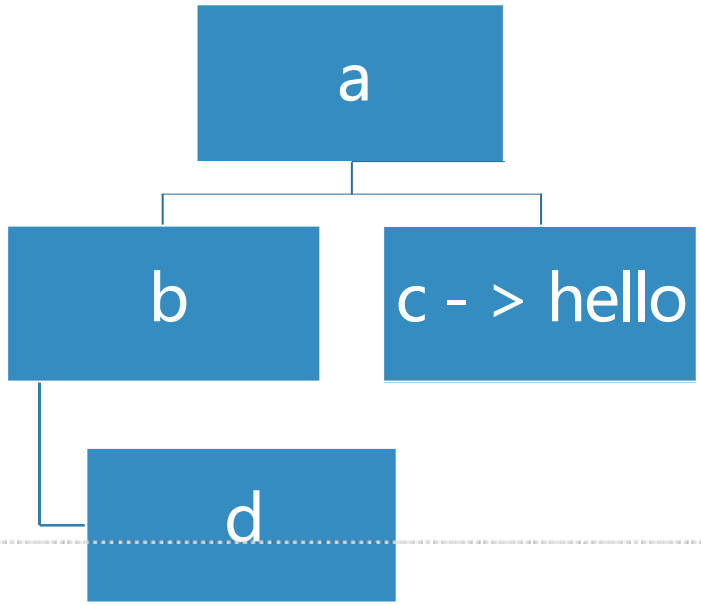
Implement the tree: the node-linking method
实现树：节点链接法

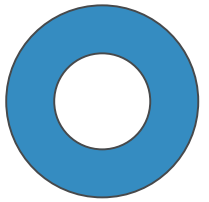
```
def getRightChild(self):
    return self.rightChild

def getLeftChild(self):
    return self.leftChild

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

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

Implement the tree: the node-linking method
实现树：节点链接法

Please draw the illustration of r
请画出r的图示

```
r = BinaryTree('a')
r.insertLeft('b')
r.insertRight('c')
r.getRightChild().setRootVal('hello')
r.getLeftChild().insertRight('d')
```





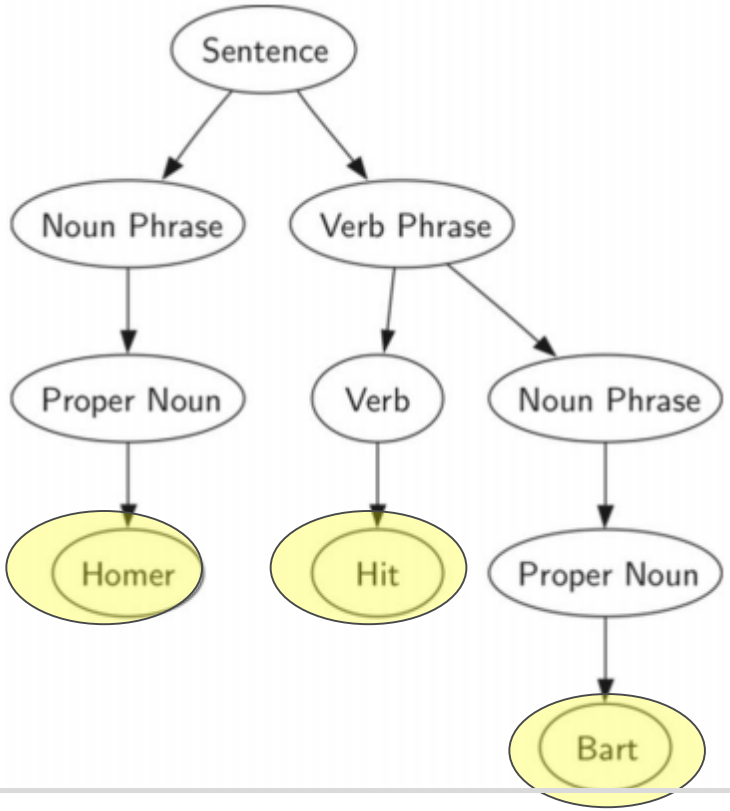
Application of Trees: Expression Parsing
树的应用：表达式解析

Application of the tree: Parsing the tree (the syntax tree)
树的应用：解析树(语法树)

By using trees to represent sentences in a language, we can analyze various grammatical components of sentences and process various components of sentences
将树用于表示语言中句子，可以分析句子的各种语法成分，对句子的各种成分进行处理

Syntax analysis tree
语法分析树

Subject，verb，object, attributive，adverbial, complement
主谓宾，定状补



Application of the tree: Parsing the tree (the syntax tree)
树的应用：解析树(语法树)

Compilation of a programming language
程序设计语言的编译

- lexical, grammar check
  词法、语法检查
- Generate the target code from the syntax tree
  从语法树生成目标代码

natural language processing
自然语言处理

- Machine translation, semantic understanding
  机器翻译、语义理解

Tree application: Expression parsing
树的应用：表达式解析

We can also represent the expression as a tree structure
我们还可以将表达式表示为树结构

Leaf node is saved as operand, internal node is saved as operator
叶节点保存操作数，内部节点保存操作符

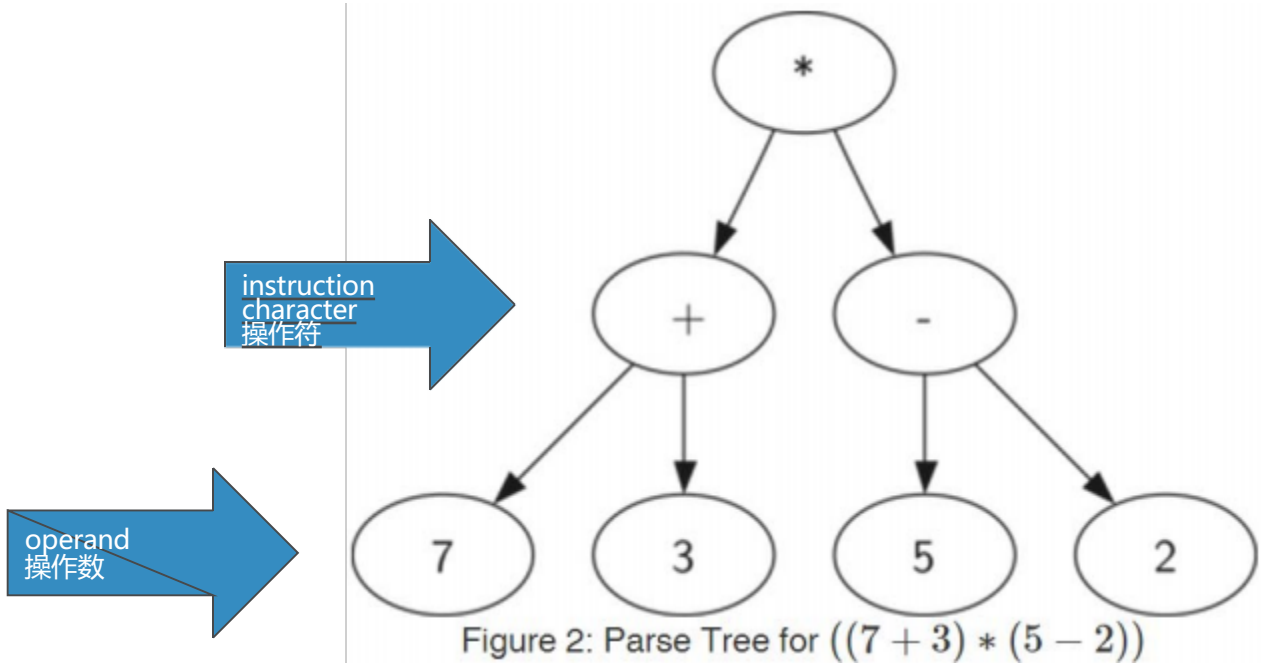


Figure 2: Parse Tree for $((7+3)*(5-2))$

Tree application: Expression parsing
树的应用：表达式解析

Full Parenthesis expression ((7 + 3) * (5-2))
全括号表达式((7+3)*(5-2))

Because of the parentheses, if you need to calculate *, you have to calculate 7 + 3 and 5-2 fisrtly, the expression hierarchy determines the priority of the lower the expression, the higher the priority

由于括号的存在，需要计算*的话，就必须先计算7+3和5-2，表达式层次决定计算的优先级越底层的表达式，优先级越高

low priority
低优先级

high priority
高优先级

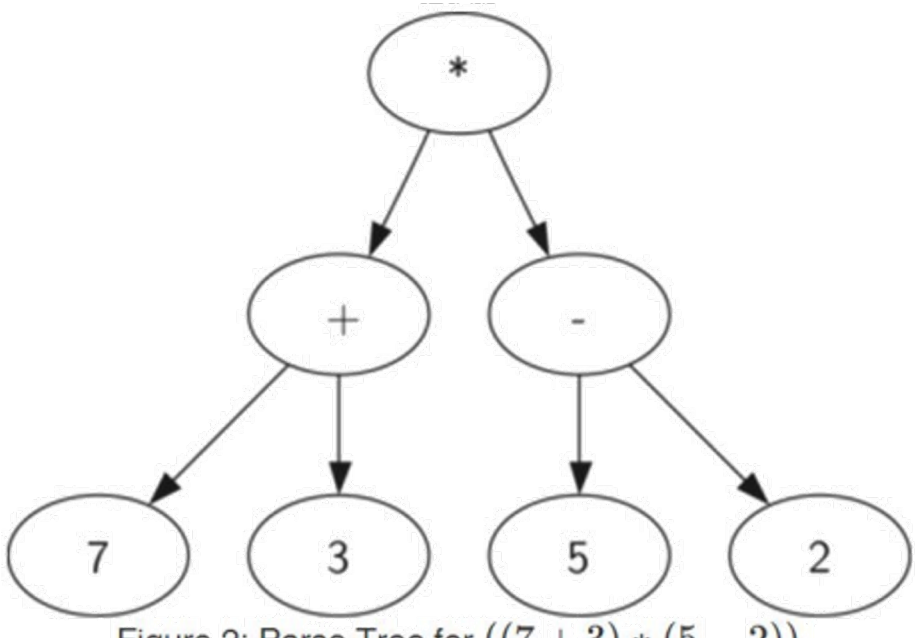


Figure 2: Parse Tree for $((7+3)*(5-2))$

Tree application: Expression parsing
树的应用：表达式解析

Each subtree in the tree represents a sub-expression
树中每个子树都表示一个子表达式

It is achieved by replacing the subtree with the node of the subexpression value

将子树替换为子表达式值的节点，即可实现求值

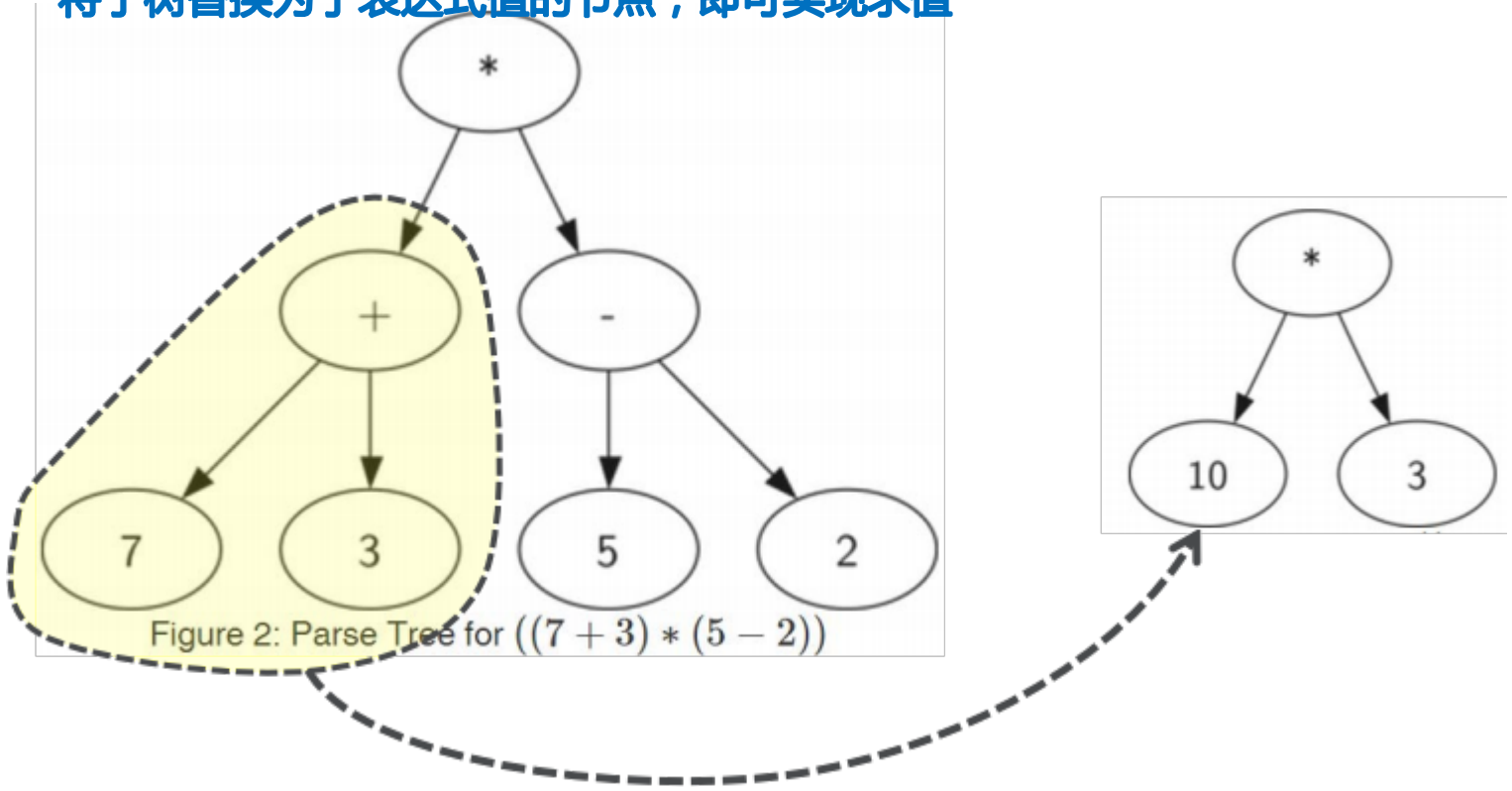


Figure 2: Parse Tree for $((7+3)*(5-2))$

Expression parsing
表达式解析

Below, we use the tree structure to try the following
下面，我们用树结构来做如下尝试

- Build a parsing tree from full parenthesis expression
  从全括号表达式构建表达式解析树
- Expression are evaluated by using an expression parsing tree
  利用表达式解析树对表达式求值
- Recovery the string form of the original expression from the expression parsing tree
  从表达式解析树恢复原表达式的字符串形式

First, the full parenthesis expression should be decomposed into word Token lists
首先，全括号表达式要分解为单词Token列表

- The words are divided into parentheses "()", operator "+ - * /", and operand "0~9"
  其单词分为括号“()”、操作符“+-*/”和操作数“0~9”这几类
- The left bracket is the beginning of the expression, and the right bracket is the end of the expression
  左括号就是表达式的开始，而右括号是表达式的结束

Establish an expression parsing tree: an instance
建立表达式解析树：实例

Full bracket expression: (3 + (4 * 5))
全括号表达式：(3+(4*5))

Decomposition into word tables
分解为单词表

```
['(','3','+','(','4','*','5',')',')']
```

(3+(4*5))

```
['(', '3', '+', '(', '4', '*', '5', ')', ')']
```

Establish an expression parsing tree: an instance
建立表达式解析树：实例

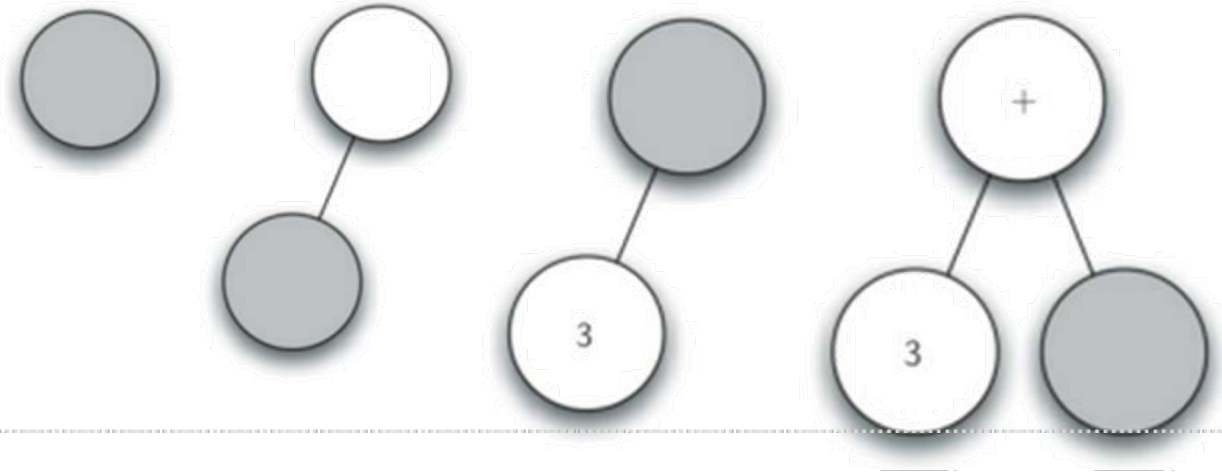
Create an expression parsing tree procedure
创建表达式解析树过程

Create an empty tree with the current node as the root node
创建空树，当前节点为根节点

Read '(', left child created, current node down
读入'('，创建了左子节点，当前节点下降

Read '3', the current node set to 3, rising to the parent
读入'3'，当前节点设置为3，上升到父节点

Read '+', the current node set to +, create right child, current node down
读入'+'，当前节点设置为+，创建右子节点，当前节点下降



Establish an expression parsing tree: an instance
建立表达式解析树：实例

Create an expression parsing tree procedure
创建表达式解析树过程

Read in '(', create the left child node, and the current node down

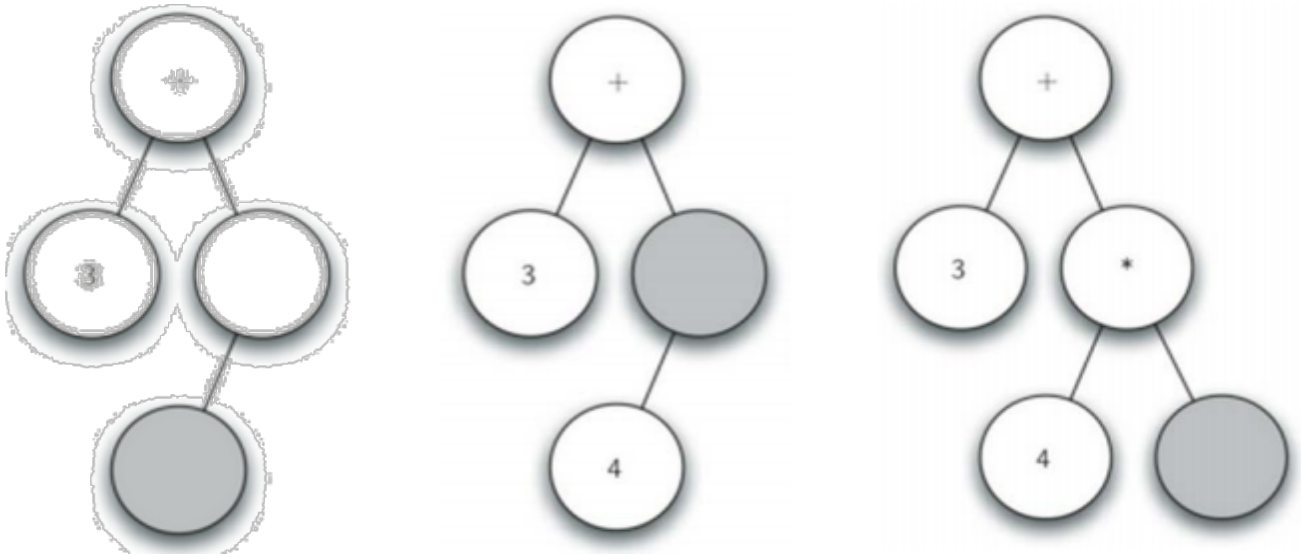
读入'('，创建左子节点，当前节点下降

Read '4', the current node is set to 4 and rises to the parent node

读入'4'，当前节点设置为4，上升到父节点

Read '*', the current node set to *, create right child, current node down

读入'*'，当前节点设置为*，创建右子节点，当前节点下降



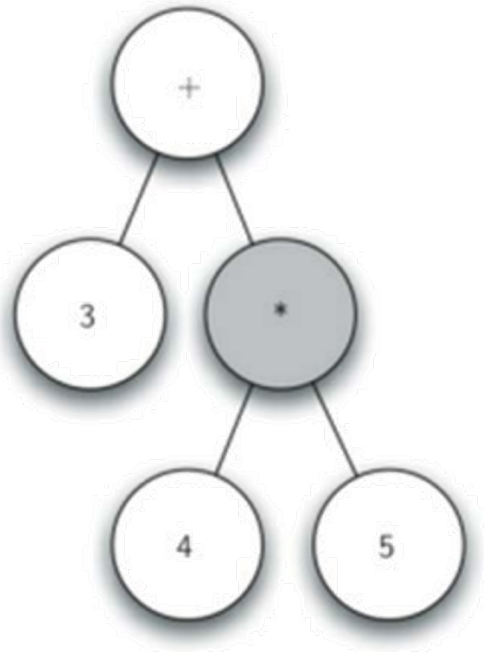
Establish an expression parsing tree: an instance
建立表达式解析树：实例

Create an expression parsing tree procedure
创建表达式解析树过程

Read '5' with the current node set to 5 and rises to the parent node
读入'5'，当前节点设置为5，上升到父节点

Read ')', and rise up to the parent node
读入')'，上升到父节点

Read ')' and rise up to the parent node
读入')'，再上升到父节点



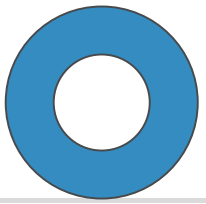
Establish an expression parsing tree: Rule
建立表达式解析树：规则

Scan for each word of the full parenthesis expression from left to right,
从左到右扫描全括号表达式的每个单词，

Establish the parsing tree by the rules： 依据规则建立解析树:

①If the current word is "(": add a new node to the current node as its left child, the current node drops down to this new node if the current word is the operator "+, -, /, *": Set the value of the current node for this symbol, add a new node to the current node as its right child node, the current node drops down to be this new node

如果当前单词是"("：为当前节点添加一个新节点作为其左子节点，当前节点下降为这个新节点如果当前单词是操作符"+,-,/,*"：将当前节点的值设为此符号，为当前节点添加一个新节点作为其右子节点，当前节点下降为这个新节点



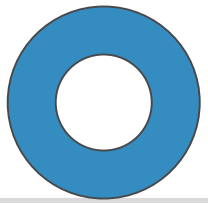
Establish an expression parsing tree: Rule
建立表达式解析树：规则

Scan for each word of the full parenthesis expression from left to right,
从左到右扫描全括号表达式的每个单词，

Establish the parsing tree by the rules 依据规则建立解析树

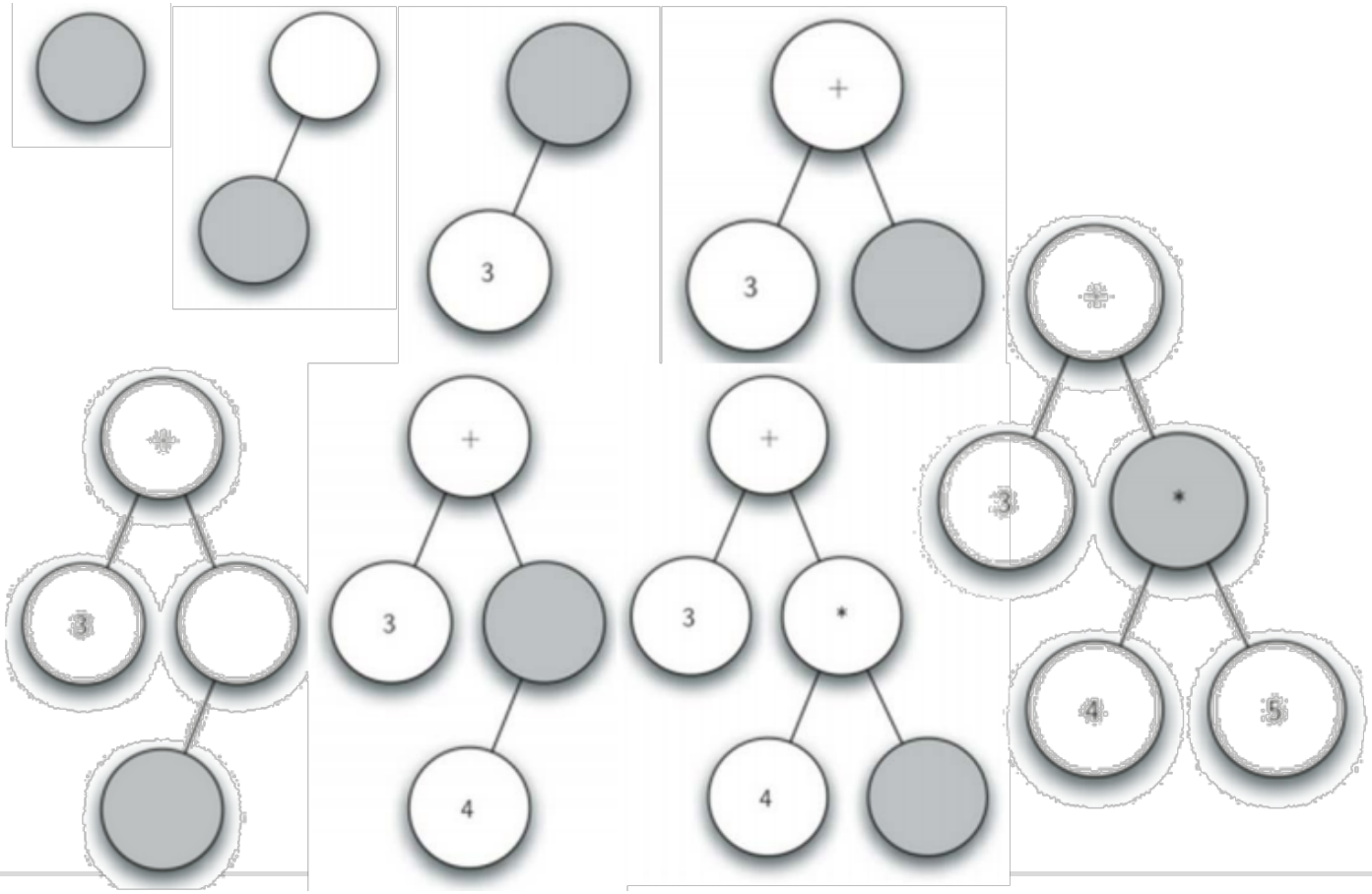
②If the current word is an operand: set the value of the current node to this number, the current node rises to the parent node
如果当前单词是操作数：将当前节点的值设为此数，当前节点上升到父节点

③If the current word is ")": then the current node rises to the parent node
如果当前单词是")"：则当前节点上升到父节点



Building Expression Parsing Trees: Examples
建立表达式解析树：实例

Full bracket expression: (3+(4*5))
全括号表达式：（3+(4*5)）



Building Expression Parsing Trees: Ideas
建立表达式解析树：思路

From the illustration process, we can see that the key to the process of creating a tree is to track the current node

从图示过程中我们看到 ，创建树过程中关键的是对当前节点的跟踪

- To create left and right subtrees call insertLeft/Right

  创建左右子树可调用insertLeft/Right

- Setting the value of the current node, you can call setRootVal

  当前节点设置值，可以调用setRootVal

- Descending to the left and right subtrees can call getLeft/RightChild

  下降到左右子树可调用getLeft/RightChild

- However, up to the parent node, this has no method support!

  但是，上升到父节点，这个没有方法支持！

We can use a stack to keep track of the parent node

我们可以用一个栈来记录跟踪父节点

- When the current node drops down, push the node before dropping into the stack

  当前节点下降时，将下降前的节点push入栈

- When the current node needs to be raised to the parent node, it can be raised to the node that pops out of the stack!

  当前节点需要上升到父节点时，上升到pop出栈的节点即可！

Building an Expression Parsing Tree: Code

建立表达式解析树：代码

```
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(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
```

push down
入栈下降

expression starts
表达式开始

push down
入栈下降

operand
操作数

pop up
出栈上升

operator
操作符

expression ends
表达式结束

pop up
出栈上升

Evaluating with Expression Parse Trees: Ideas

利用表达式解析树求值： 思路

An expression parse tree is created, which can be used for evaluation. Since the binary tree is a recursive data structure, it can naturally be processed by a recursive algorithm. Evaluate the recursive function *evaluate*

创建了表达式解析树 ，可用来进行求值,由于二叉树是一个递归数据结构，自然可以用递归算法来处理,求值递归函数 evaluate

From the above description of the subexpression, we can start from the bottom subtree of the tree, and gradually evaluate to the upper layer, and finally get the value of the entire expression

由前述对子表达式的描述，可从树的底层子树开始，逐步向上层求值，最终得到整个表达式的值

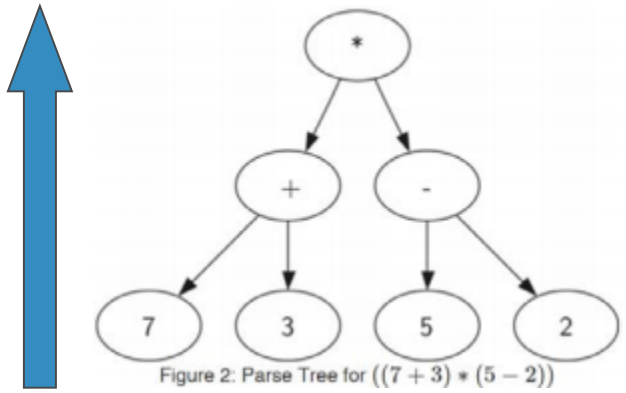


Figure 2: Parse Tree for $((7+3)*(5-2))$

Evaluating with Expression Parsing Trees: Ideas

利用表达式解析树求值： 思路

The recursive three elements of the evaluation function evaluate:

求值函数evaluate的递归三要素：

Basic end condition: leaf node is the simplest subtree,with no left right child node, the data item of its root node is the value of the subexpression tree

基本结束条件：叶节点是最简单的子树，没有左右子节点，其根节点的数据项即为子表达式树的值

Downsizing: Divide the expression tree into left subtree and right subtree, that is, reduce the scale

缩小规模：将表达式树分为左子树、右子树，即为缩小规模

Call itself: call *evaluate* to calculate the values of the left subtree and the right subtree respectively, and then calculate the values of the left and right subtrees according to the operator of the root node, so as to obtain the value of the expression

调用自身：分别调用evaluate计算左子树和右子树的值，然后将左右子树的值依根节点的操作符进行计算，从而得到表达式的值

Evaluating with Expression Parsing Trees: Ideas

利用表达式解析树求值： 思路

A trick to increase program readability: function references

一个增加程序可读性的技巧：函数引用

import operator

op= operator.add

```
>>> import operator
>>> operator.add
<built-in function add>
>>> operator.add(1,2)
3
>>> op= operator.add
>>> n= op(1,2)
>>> n
3
```

Evaluate with expression parsing tree: code

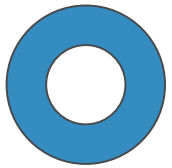
利用表达式解析树求值：代码

```
import operator
def evaluate(parseTree):
    opers = {'+':operator.add, '-':operator.sub, \
             '*':operator.mul, '/':operator.truediv}

    leftC = parseTree.getLeftChild()
    rightC = parseTree.getRightChild()

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

downsize
缩小规模

recursive call
递归调用

basic end condition
基本结束条件



tree traversal
树的遍历

Tree Traversals
树的遍历

The operation of accessing all data items in a dataset is called "Traversal"
对一个数据集中的所有数据项进行访问的操作称为“遍历”

In a linear data structure, access to all its data items is relatively straightforward
线性数据结构中，对其所有数据项的访问比较简单直接

- Just do it in order
  按照顺序依次进行即可

The nonlinear characteristics of the tree make the traversal operation more complicated
树的非线性特点，使得遍历操作较为复杂

Tree Traversals
树的遍历

We distinguish three types of traversal according to the order of access to nodes

我们按照对节点访问次序的不同来区分3种遍历

①Preorder traversal (preorder): first visit the root node, then recursively visit the left subtree, and finally visit the right subtree in preorder;

前序遍历(preorder)：先访问根节点，再递归地前序访问左子树、最后前序访问右子树；

②Inorder traversal (inorder): first recursively visit the left subtree in inorder, then visit the root node, and finally visit the right subtree in inorder;

中序遍历(inorder)：先递归地中序访问左子树，再访问根节点，最后中序访问右子树；

③Postorder traversal (postorder): first recursively visit the left subtree, then postorderly visit the right subtree, and finally access the root node.

后序遍历(postorder)：先递归地后序访问左子树，再后序访问右子树，最后访问根节点。

Example of preorder traversal: reading chapters of a book

前序遍历的例子：一本书的章节阅读

Book->Ch1->S1.1->S1.2->S1.2.1->S1.2.2-
>Ch2->S2.1->S2.2->S2.2.1->S2.2.2

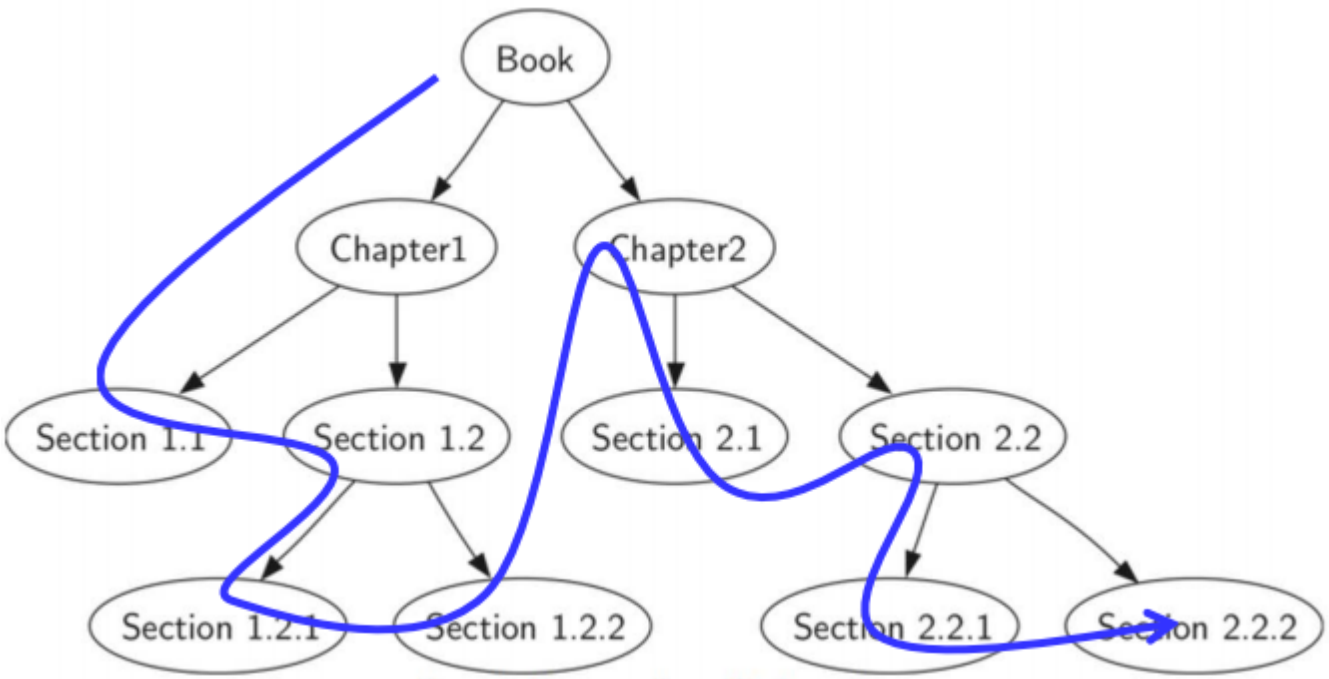


Figure 5: Representing a Book as a Tree

Tree Traversal: Recursive Algorithm Code
树的遍历：递归算法代码

The code for tree traversal is pretty neat!
树遍历的代码非常简洁！

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

The code for post-order and in-order traversal only needs to be reordered
后序和中序遍历的代码仅需要调整顺序

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

Tree Traversal: Recursive Algorithm Code
树的遍历：递归算法代码

The method of preorder traversal can also be implemented in the BinaryTree class:
也可以在BinaryTree类中实现前序遍历的方法：

Need to join the judgment of whether the subtree is empty
需要加入子树是否为空的判断

```
def preorder(self):
    print(self.key)
    if self.leftChild:
        self.leftChild.preorder()
    if self.rightChild:
        self.rightChild.preorder()
```

Postorder Traversal: Expression Evaluation

后序遍历：表达式求值

Looking back at the forementioned expression parsing tree evaluation, it is actually a post-order traversal process.

回顾前述的表达式解析树求值，实际上也是一个后序遍历的过程

Rewrite the expression evaluation code using postorder traversal:

采用后序遍历法重写表达式求值代码：

```
def postordereval(tree):
    opers = {'+':operator.add, '-':operator.sub, \
             '*':operator.mul, '/':operator.truediv}
    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()
```

left subtree
左子树

right subtree
右子树

Inorder traversal: generate full bracketed infix expressions

中序遍历：生成全括号中缀表达式

Use inorder traversal recursive algorithm to generate full bracket infix expressions

采用中序遍历递归算法来生成全括号中缀表达式

In the following code, parentheses are also added to each number, please modify the code to remove it by yourself (after-school exercise)

下列代码中对每个数字也加了括号，请自行修改代码去除(课后练习)

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