数据结构和算法
(Python描述)

郭 炜

微信公众号

微博：http://weibo.com/guoweiofpku

学会程序和算法，走遍天下都不怕!

讲义照片均为郭炜拍摄

二叉树

二叉树概念及性质

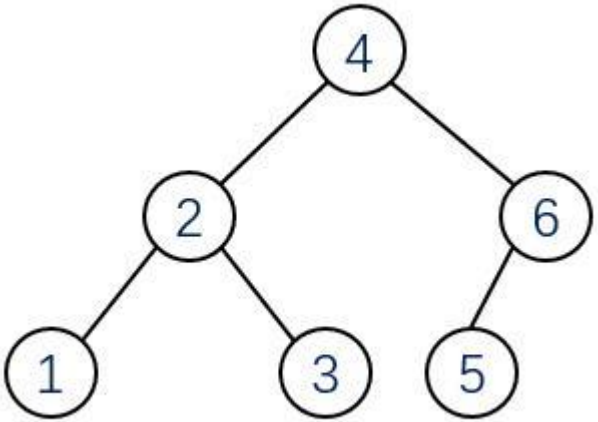
黄山

二叉树的定义

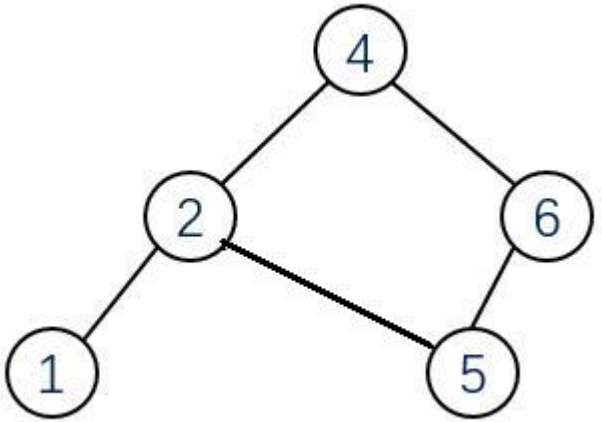
1) 节点：由三部分组成：数据、左子节点指针、右子节点指针
2) 一个左右子节点指针均为空的节点，叫叶子节点
3) 叶子节点是一棵二叉树，树根即是该节点
4) 若有一个节点X的左子节点指针或右子节点指针指向一棵不包含X的二叉树的根，或两者分别指向两棵没有公共节点且不包含X的二叉树的根，则X和其指向的一棵或两棵二叉树构成一棵二叉树，根为X。X的左子树是左子节点指向的树，右子树是右子节点指向的树

二叉树的定义

二叉树

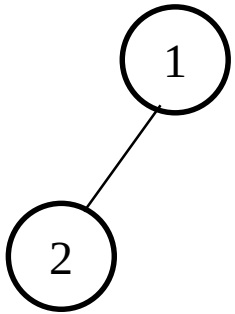


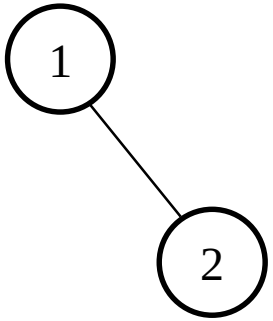
非二叉树，因不满足没有公共节点条件



二叉树的定义

- 二叉树的左右子树是有区别的，以下是两棵不同的二叉树:





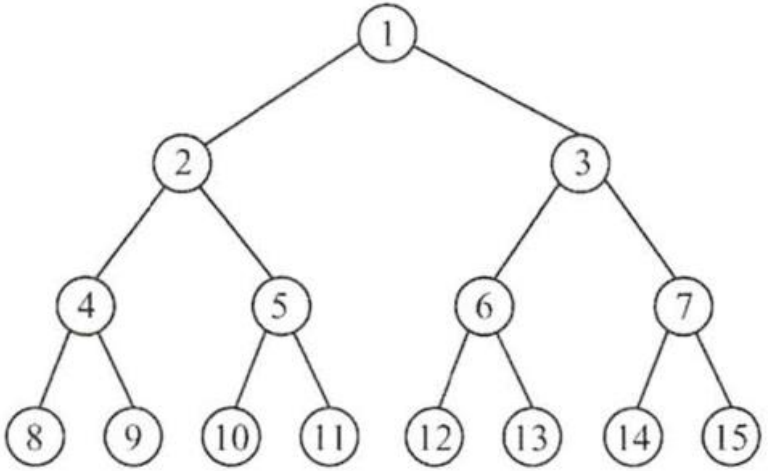
二叉树相关的概念

- 父节点、祖先节点
  如果a是b的子节点，则b是a的父节点
  父节点是祖先节点。祖先节点的祖先节点也是祖先节点。
- 度
  节点的子树的个数
- 树的边
  连接父节点和子节点的指针
- 节点的层次
  从根出发到达节点所经过的边数。根节点为第0层
- 树的高度
  即节点层数，为节点最大层次+1。树的高度是>=1的

二叉树相关的概念

- 满二叉树

每一层节点数目都达到最大。即第i层有2^i个节点。高为h的满二叉树，有2^h -1个节点

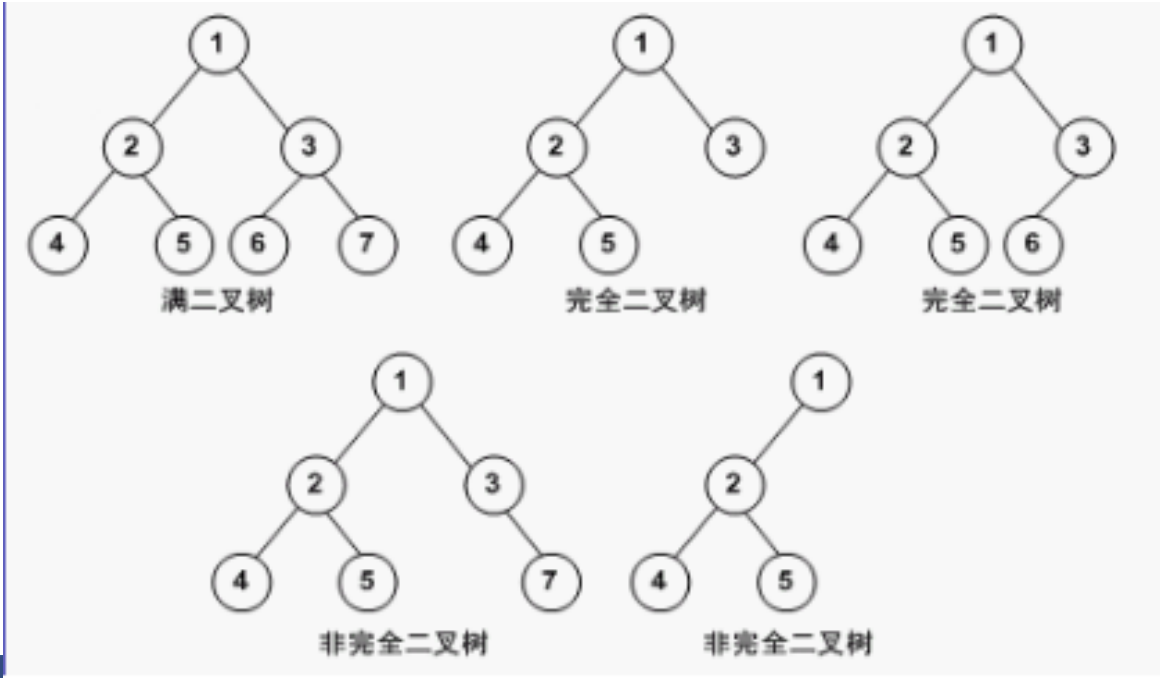


满二叉树

二叉树相关的概念

- 完全二叉树

  除最后一层外，其余层的节点数目均达到最大。而且，最后一层节点若不满，则缺的节点定是在最右边的连续若干个



二叉树的性质

福建宁德三都澳鱼排

二叉树的性质

- 1) 第i层最个多2^i个节点。高为h的满二叉树节点总数2^h-1

- 2) 节点数为n的树，边的数目为n-1
- 3) 包含n个结点的二叉树的高度至少为$\log_2 (n+1)$向上取整
- 4) 在任意一棵二叉树中，若叶子节点的个数为n_0，度为2的节点个数为n_2，则$n_0=n_2+1$。
- 5) 任何两个节点之间只有一条路径

2),4)按树的高度用数学归纳法证明

完全二叉树的性质

- 完全二叉树

  除最后一层外，其余层的节点数目均达到最大。而且，最后一层节点若不满，则缺的节点定是在最右边的连续若干个

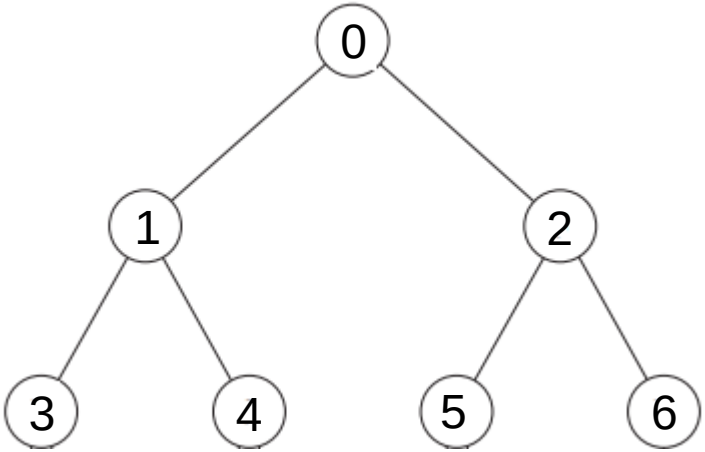
  节点数目为n的完全二叉树，高度为为$\log_2 (n+1)$向上取整

  可以用列表存放完全二叉树的节点，不需要左右子节点指针。下标为i的节点的左子节点下标是2*i+1,右子节点是2*i+2

完全二叉树的性质

- 完全二叉树

  可以用列表存放完全二叉树的节点，不需要左右子节点指针。下标为i的节点的左子节点下标是2*i+1,右子节点是2*i+2



二叉树的实现

山西应县木塔

二叉树的实现方法

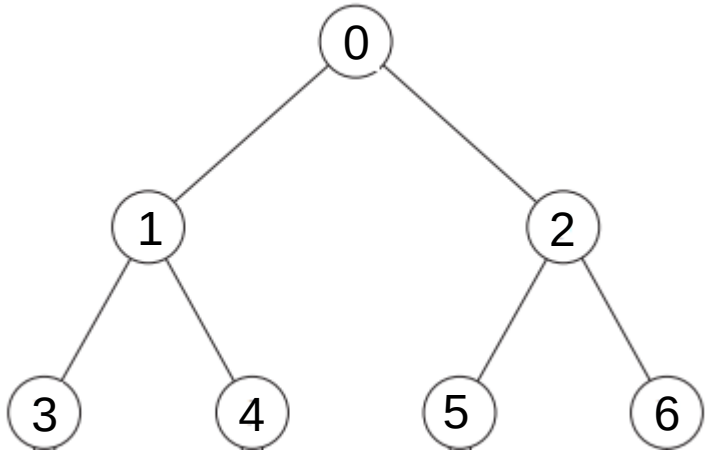
```
class BinaryTree:
        def __init__(self,data,left = None,right = None):
                self.data,self.left,self.right = data,left,right
        def addLeft(self,tree): #tree是一个二叉树
                self.left = tree
        def addRight(self,tree): #tree是一个二叉树
                self.right = tree
```

二叉树的列表实现方法

- 二叉树是一个三个元素的列表X
- X[0]是根节点的数据，X[1]是左子树，X[2]是右子树。如果没有左子树，X[1]就是空表[]，如果没有右子树，X[2]就是空表。
- 叶子节点为：[data,[],[]]

二叉树的列表实现方法

```
[0,
    [1,
      [3,[],[]],
      [4,[],[]] ],
    [2,
      [5,[],[]],
      [6,[],[]] ]
]
```



即: [0, [1, [3, [], []], [4, [], []]], [2, [5, [], []], [6, [], []]]]

二叉树的列表实现方法

```
class BinaryTree:
      def __init__(self,data,left = None,right = None):
            self.treeList = [data,left,right]
      def addLeft(self,tree):
            self.treeList[1] = tree.treeList
      def addRight(self,tree):
            self.treeList[2] = tree.treeList
```

例题：构建二叉树

读入：

A

B

0

D

E

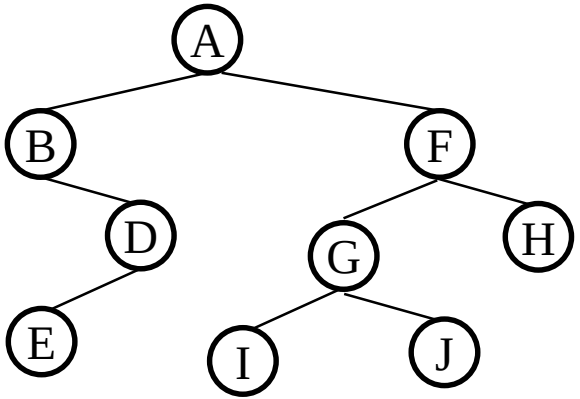
F

G

I

J

H



构建二叉树。0表示没有左子树

例题：构建二叉树

```
def buildTree(level): #读取nodesPtr指向的那一行，并建立以其为根的子树
    #该根的层次是level。建好后，令nodesPtr指向该子树的下一行
    global nodesPtr
    tree = BinaryTree(nodes[nodesPtr][1]) #建根节点
    nodesPtr += 1 #看下一行
    if nodesPtr < len(nodes) and nodes[nodesPtr][0] == level + 1:
        if nodes[nodesPtr][1] != '0':
            tree.addLeft (buildTree(level + 1))
        else:
            nodesPtr += 1
    if nodesPtr < len(nodes) and nodes[nodesPtr][0] == level + 1:
        tree.addRight(buildTree(level + 1))
    return tree
```

例题：构建二叉树

```
nodes = []
while True:
    try:
        s = input().rstrip()
        nodes.append((len(s)-1,s.strip()))
    except:
        break
nodesPtr = 0   #表示看到nodes里的第几行
tree = buildTree(0)
```

nodes内容形如：

```
[(0, 'A'), (1, 'B'), (2, '0'), (2, 'D'), (3, 'E'), (1, 'F'), (2, 'G'),
(3, 'I'), (3, 'J'), (2, 'H')]
```

元素为(缩进，数据)

二叉树的遍历


日本奈良东大寺

二叉树的遍历

- 广度优先遍历：使用队列
- 深度优先遍历：编写递归函数
  1) 前序遍历：先处理根节点，再遍历左子树，再遍历右子树
  2) 中序遍历：先遍历左子树，再处理根节点，再遍历右子树
  3) 后序遍历：先遍历左子树，再遍历右子树，再处理根节点
- 遍历只需要访问每个节点一次，因此复杂度O(n)。n是总节点数目。

二叉树的前序遍历

```
def preorderTravel(tree,op): #前序遍历
    #tree是一个BinaryTree对象,op是操作
    op(tree.data)
    if tree.left:
        preorderTravel(tree.left,op)
    if tree.right:
        preorderTravel(tree.right,op)
```

用法如:

```
preorderTravel(tree,lambda x:print(x,end=""))
```

二叉树的前序遍历

写成BinaryTree类的方法：

```
    def preorderTraveler(self,root,op):
        op(root.data)
        if root.left:
            self.preorderTraveler(root.left,op)
        if root.right:
            self.preorderTraveler(root.right,op)
    def preorderTravel(self,op):
        self.preorderTraveler(self,op)
```

用法如：

```
tree.preorderTravel(lambda x:print(x,end="")
#tree是一个BinaryTree对象
```

二叉树的中序遍历

```
def inorderTravel(tree, op): #中序遍历
    # tree是一个BinaryTree对象,op是操作
    if tree.left:
        inorderTravel(tree.left, op)
    op(tree.data)
    if tree.right:
        inorderTravel(tree.right, op)
```

用法如:

```
inorderTravel(tree,lambda x:print(x,end=""))
```

二叉树的后序遍历

```
def postorderTravel(tree, op): #后序遍历
    # tree是一个BinaryTree对象,op是操作
    if tree.left:
        postorderTravel(tree.left, op)
    if tree.right:
        postorderTravel(tree.right, op)
    op(tree.data)
```

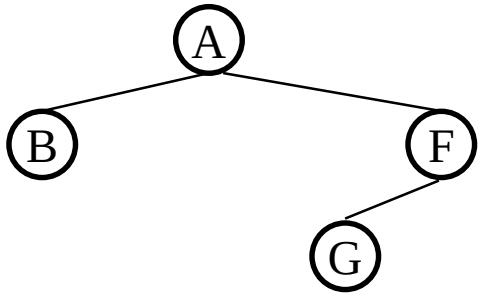
用法如:

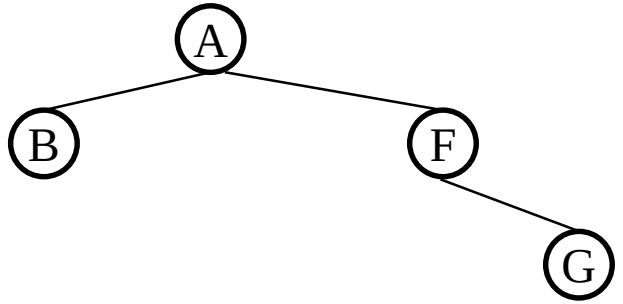
```
postorderTravel(tree,lambda x:print(x,end=""))
```

遍历序列和二叉树

1．仅凭一种遍历序列(前序、后序、中序)，不能确定二叉树的样子

2．给出一棵二叉树的前序遍历序列，和后序遍历序列，依然不能确定这棵树的样子





上面两棵二叉树有相同前序序列和中序序列

遍历序列和二叉树

给出一棵二叉树的中序遍历序列，再加上前序序列，或后序序列，就可以确定树的样子

由前序序列和中序序列构造二叉树，假设序列分别为于列表P和列表Q

1） P[0]是树的树根

2）找到树根P[0]在中序序列中的位置x，并将中序序列以树根为界分为左子树的中序序列Q[:X]和右子树的中序序列Q[x+1:]

3）P[1:X+1]是左子树的前序序列，P[X+1:]是右子树的前序序列，递归建两棵子树

用生成器遍历二叉树

用生成器遍历二叉树

需求: 假设tree是一棵二叉树，希望能以for循环的形式访问tree的前序序列，并且随时可以break，且不希望事先生成整个前序序列

```
#按前序遍历的顺序，输出tree中的元素，碰到元素100就停止
for x in tree.preorderSeq():
    print(x)
    if x == 100:
        break
```

用生成器，加上前序遍历的非递归函数来解决！

生成器(generator)

- yield关键字用来定义生成器（Generator），可以当return使用，从函数里返回一个值

- 使用了 yield 的函数被称为生成器（generator）。当函数被调用的时候，并不执行函数，而是返回一个迭代器(iterator)

- 如果 X 是一个生成器被调用时的返回值（迭代器），则

```
for i in X:
    print(i)
```

会依次打印X中yield语句返回的结果，直到函数X再也不会执行到yield语句

生成器

```
def test_yield(): #调用则返回迭代器
    yield 1
    yield 2
    yield (1,2)
for x in test_yield():
   print(x)
```

输出:

```
1
2
(1,2)
```

生成器

- 使用 yield 实现斐波那契数列：

```
def fibonacci(n): # 生成器函数 - 用于求斐波那契数列前n项
    a, b, counter = 0, 1, 0
    while counter <= n:
        yield a
        a, b = b, a + b
        counter += 1

for x in fib(10):
   print(x,end = ",")
```

0 1 1 2 3 5 8 13 21 34 55

迭代器+生成器的一个优点就是它不要求事先准备好整个迭代过程中所有的元素。迭代器仅仅在迭代至某个元素时才计算该元素，而在这之前或之后，元素可以不存在或者被销毁。这个特点使得它特别适合用于遍历一些巨大的或是无限的集合，比如几个G的文件，或是斐波那契数列等等。这个特点被称为延迟计算或惰性求值(Lazy evaluation)。

二叉树的非递归遍历

```
class BinaryTree:
    def __init__(self,data,left = None,right = None):
        self.data,self.left,self.right = data,left,right
    def preorderTravelEx(self,op): #非递归前序遍历
        stack = [[self,0]] #0表示self的左子树还没有遍历过
        while len(stack) > 0:
                node = stack[-1]
                if node[0] == None: #node[0]是树节点
                        stack.pop()
                        continue
                if node[1] == 0: #左子树还没有遍历过
                        op(node[0].data)#如果是生成器，则此处 yield node[0].data
                        node[1] = 1
                        stack.append([node[0].left,0])
```

二叉树的非递归遍历

```
elif node[1] == 1: #左子树已经遍历过
        node[1] = 2
        stack.append([node[0].right, 0])
else: #右子树也遍历完了
        stack.pop()
```

二叉树前序遍历生成器

```
class BinaryTree:
    def __init__(self,data,left = None,right = None):
        self.data,self.left,self.right = data,left,right
    def preorderTravelSeq(self): #另一种前序遍历的非递归写法
        stack = [self]
        while len(stack) > 0:
                node = stack.pop()
                if node == None:
                        continue
                yield node.data
                stack.append(node.right)
                stack.append(node.left)
```

用法:

```
for x in tree.preorderTravelSeq():
    print(x,end="")
```

二叉树的应用

实例：表达式树

前序遍历得到前缀表达式：

- + 3 * 2 9 / 6 4

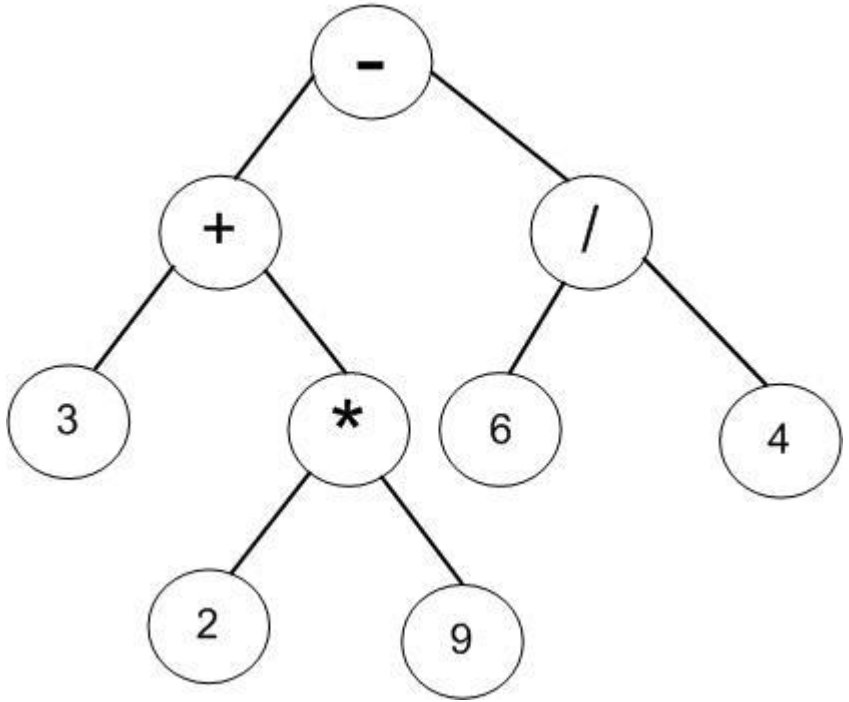
后序遍历得到后缀表达式：

3 2 9 * + 6 4 / -

中序遍历得到运算符中置表达式：

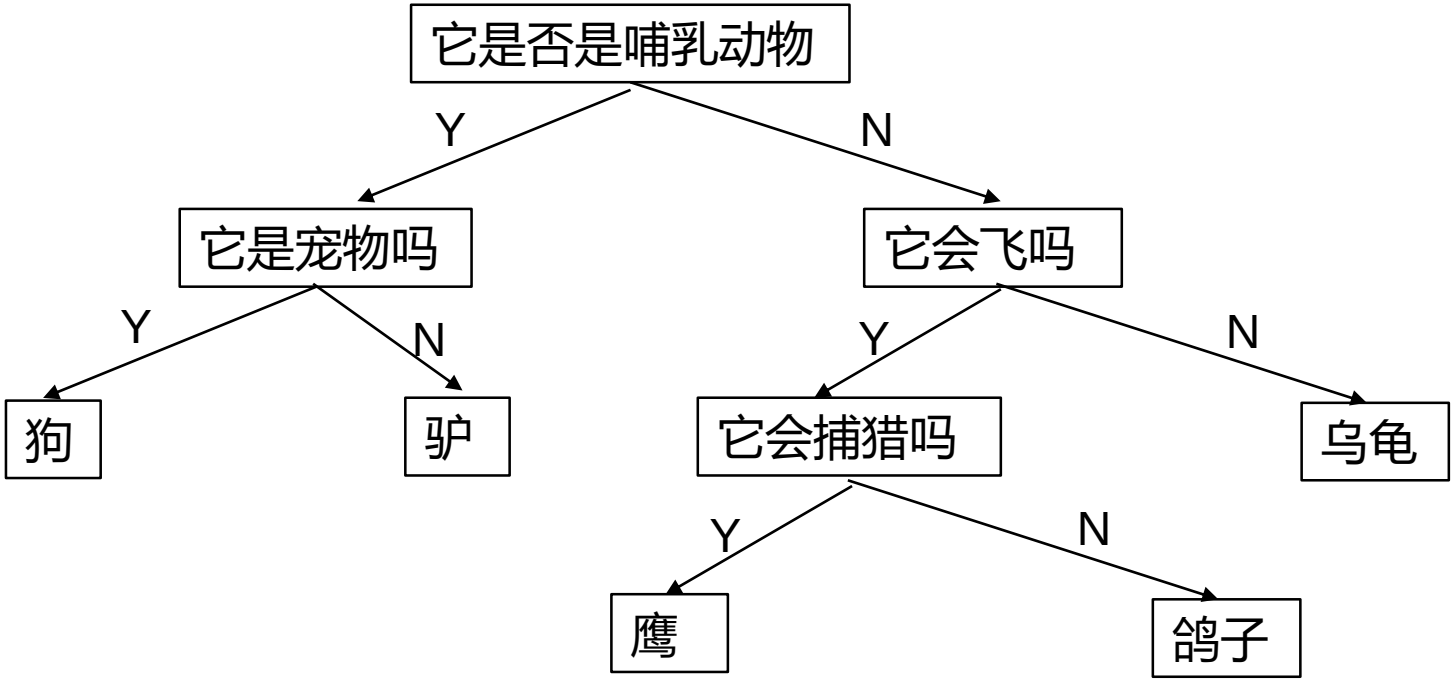
3 + 2 * 9 – 6 / 4

必要的时候要添加括号



实例：动物分类问答知识树

一个存储了狗、驴、鹰、乌龟、鸽子五种动物的系统，用户想好一个动物，系统提问，用户回答是或否，系统猜出用户想的动物。



实例：哈夫曼编码树

to be continued.....