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

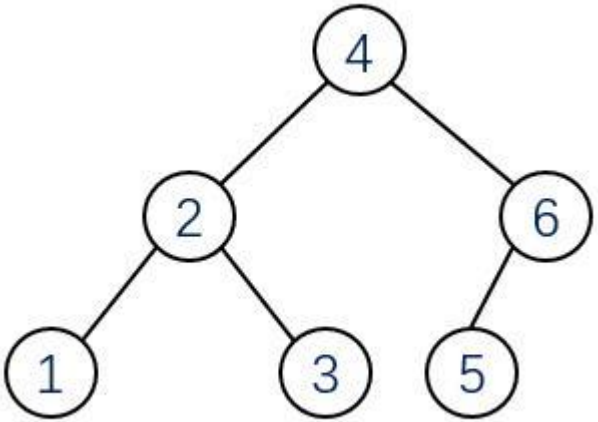
1) 二叉树是有限个元素的集合。

2) 空集合是一个二叉树，称为空二叉树。

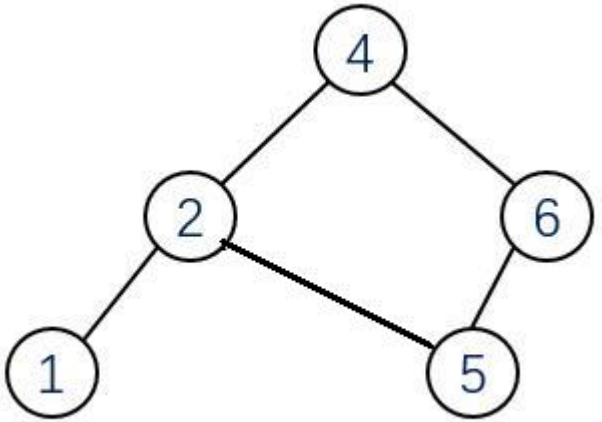
3) 一个元素(称其为“根”或“根结点”)，加上一个被称为“左子树”的二叉树，和一个被称为“右子树”的二叉树，就能形成一个新的二叉树。要求根、左子树和右子树三者没有公共元素。

二叉树的定义

二叉树

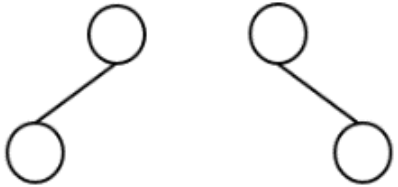


非二叉树，因不满足没有公共结点条件

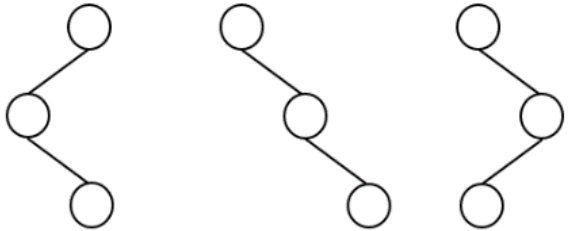


二叉树的定义

- 二叉树的左右子树是有区别的
- 以下是两棵不同的二叉树:



- 以下是三棵不同的二叉树:



二叉树的相关概念

- 二叉树的的元素称为“结点”。结点由三部分组成：数据、左子结点指针、右子结点指针。
- 结点的度(degree)：结点的非空子树数目。也可以说是结点的子结点数目。
- 叶结点(leaf node)：度为0的结点。
- 分支结点：度不为0的结点。即除叶子以外的其他结点。也叫内部结点。
- 兄弟结点(sibling)：父结点相同的两个结点，互为兄弟结点。
- 结点的层次(level)：树根是第0层的。如果一个结点是第n层的，则其子结点就是第n+1层的。
- 结点的深度(depth)：即结点的层次。

二叉树的相关概念

- 祖先(ancestor):
  1) 父结点是子结点的祖先
  2) 若a是b的祖先，b是c的祖先，则a是c的祖先。

- 子孙(descendant)：也叫后代。若结点a是结点b的祖先，则结点b就是结点a的后代。

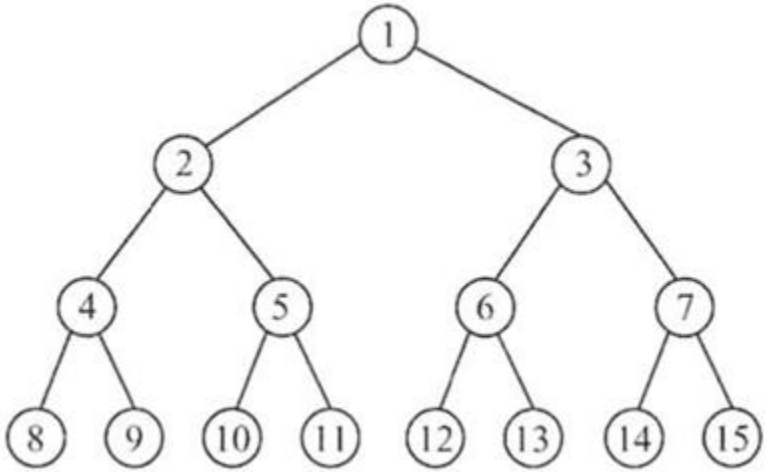
- 边：若a是b的父节点，则对子<a,b>就是a到b的边。在图上表现为连接父节点和子节点之间的线段。

- 二叉树的高度(height)：二叉树的高度就是结点的最大层次数。只有一个结点的二叉树，高度是0。结点一共有n层，高度就是n-1。

二叉树相关的概念

- 完美二叉树(perfect binary tree)

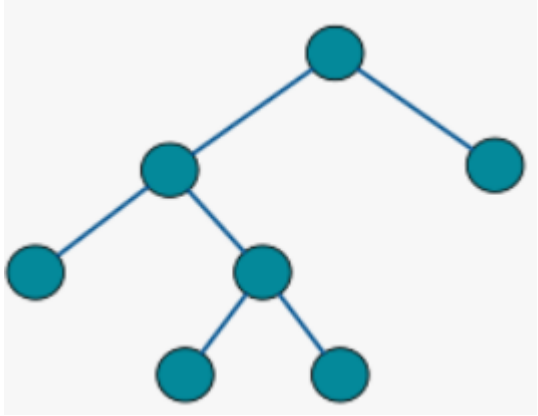
每一层结点数目都达到最大。即第i层有2^i个结点。高为h的完美二叉树，有2^h -1个结点



二叉树相关的概念

- 满二叉树（full binary tree)

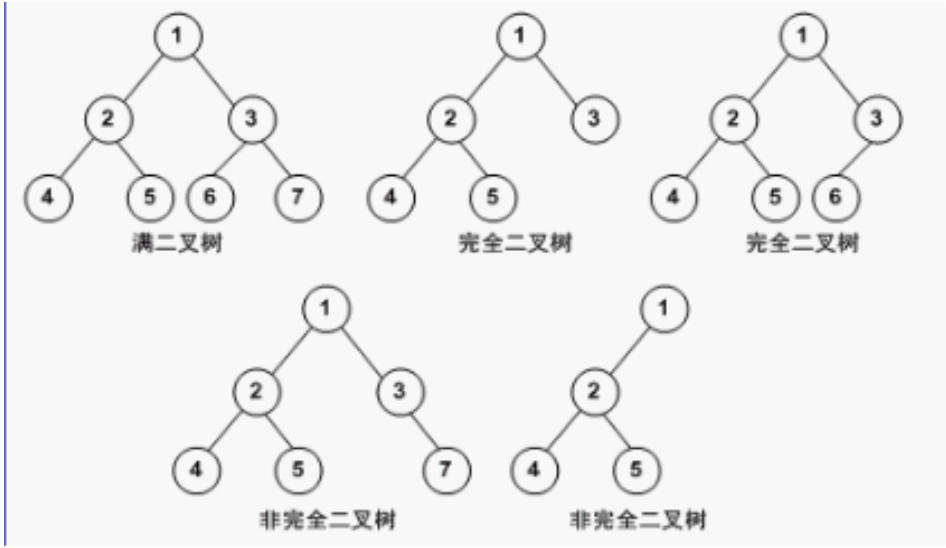
  没有1度结点的二叉树



二叉树相关的概念

- 完全二叉树(complete binary tree)

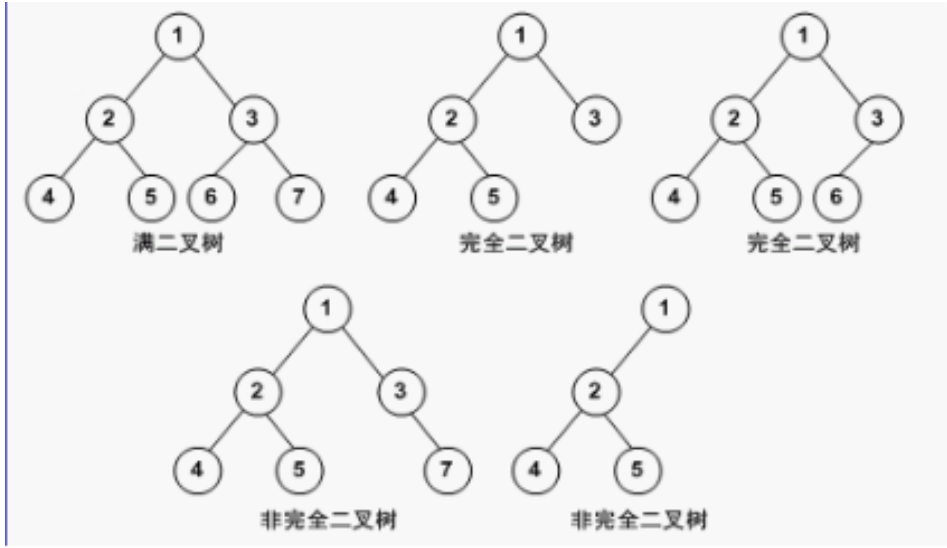
除最后一层外，其余层的结点数目均达到最大。而且，最后一层结点若不满，则缺的结点定是在最右边的连续若干个



二叉树相关的概念

- 完全二叉树(complete binary tree)

除最后一层外，其余层的结点数目均达到最大。而且，最后一层结点若不满，则缺的结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个结点
- 2) 高为h的二叉树结点总数最多$2^{h+1}-1$
- 3) 结点数为n的树，边的数目为n-1
- 4) n个结点的非空二叉树至少有$\lceil \log_2(n+1) \rceil$层结点，即高度至少为$\lceil \log_2(n+1) \rceil - 1$
- 5) 在任意一棵二叉树中，若叶子结点的个数为n0，度为2的结点个数为n2，则n0=n2+1。
- 6) 非空满二叉树叶结点数目等于分支结点数目加1。

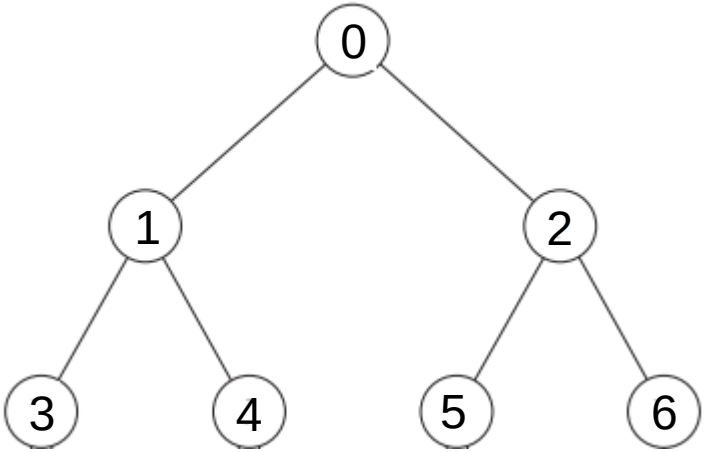
完全二叉树的性质

- 1) 完全二叉树中的1度结点数目为0个或1个
- 2) 有n个结点的完全二叉树有$\lfloor (n+1)/2 \rfloor$个叶结点。
- 3) 有n个叶结点的完全二叉树有2n或2n-1个结点(两种都可以构建)
- 4) 有n个结点的非空完全二叉树的高度为$\lfloor \log_2(n+1) \rfloor - 1$。

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结点，不需要左右子结点指针。下标为i的结点的左子结点下标是2*i+1,右子结点是2*i+2 (根下标为0)。下标为i的元素，其父结点的下标就是(i-1)//2



二叉树的实现

山西应县木塔

二叉树的实现方法

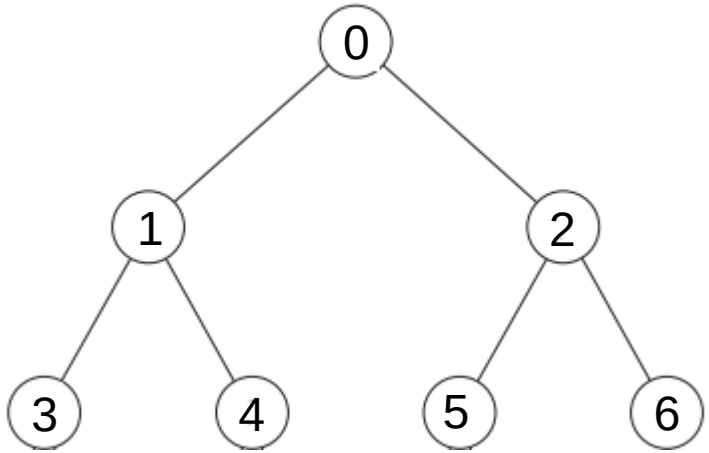
```
class BinaryTree:
        def __init__(self,data,left = None,right = None):
                self.data,self.left,self.right = data,left,right
        def addLeft(self,tree): #tree是一个二叉树
                self.left = tree
        def addRight(self,tree): #tree是一个二叉树
                self.right = tree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

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]是根结点的数据，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。
- 叶子结点为：[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
```

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，按层遍历

- 深度优先遍历：编写递归函数

  前序遍历过程：1)访问根结点 2)前序遍历左子树 3)前序遍历右子树。

  中序遍历过程：1)中序遍历左子树 2)访问根结点 3)中序遍历右子树。

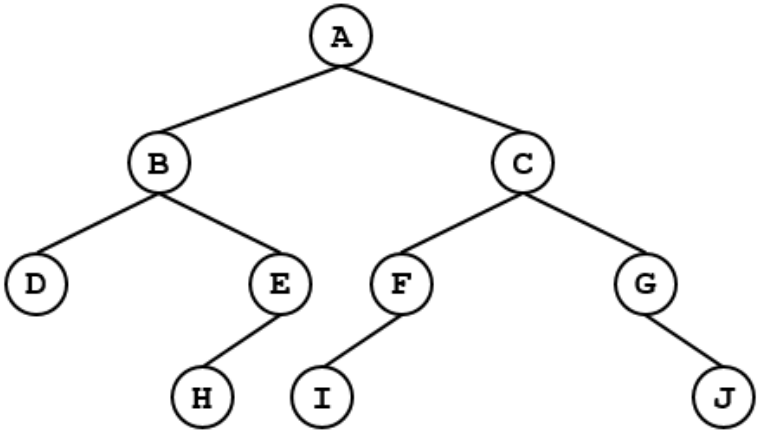
  后序遍历过程：1)后序遍历左子树 2)后序遍历右子树 3)访问根结点。

  “访问”指的是对结点进行某种具体操作，比如输出其值、修改其值等。

- 遍历只需要访问每个结点一次，因此复杂度O(n)。n是总结点数目。

二叉树的遍历

- 前序遍历访问序列：ABDEHCFIGJ
- 中序遍历访问序列：DBHEAIFCGJ
- 后续遍历访问序列：DHEBIFJGCA
- 按层遍历访问序列：ABCDEFGHIJ



二叉树的遍历

```
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
    def preorderTraversal(self, op): #前序遍历,op是函数，表示操作
        op(self)          #访问根结点
        if self.left:   #左子树不为空
            self.left.preorderTraversal(op)  #遍历左子树
        if self.right:
            self.right.preorderTraversal(op)  #遍历右子树
```

二叉树的遍历

```
    def inorderTraversal(self, op): #中序遍历
        if self.left:
            self.left.inorderTraversal( op)
        op(self)
        if self.right:
            self.right.inorderTraversal(op)
    def postorderTraversal(self, op): #后序遍历
        if self.left:
            self.left.postorderTraversal(op)
        if self.right:
            self.right.postorderTraversal(op)
        op(self)
```

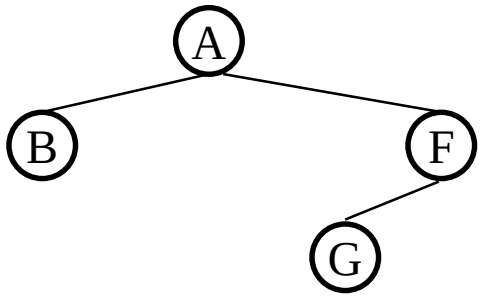
- 用法:

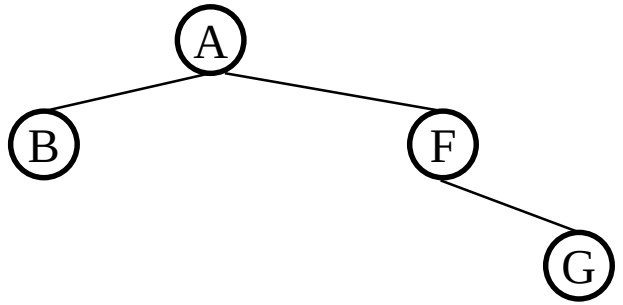
```
tree.preorderTraversal(lambda x: print(x.data,end=""))
tree.preorderTraversal(lambda x: x.data+=100)
```

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

例题：文本二叉树

文本二叉树就是由若干行文本来表示的一棵二叉树。其定义如下:

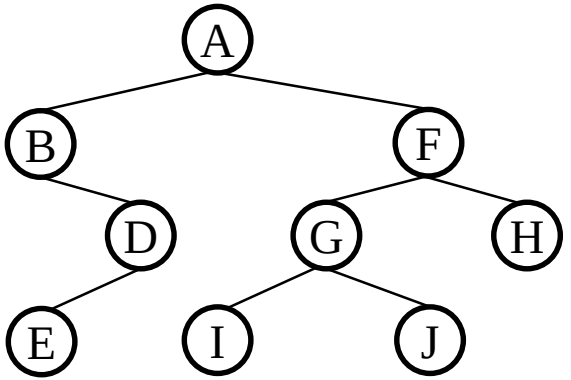
1) 每一行代表一个结点，每个结点是一个英文字母。
2) 每个结点的父结点，就是它上方，离它最近的，比它往左偏移了一个制表符的那个结点。没有父结点的结点，是树根。
3) 如果一个结点的左子树为空，则在其下面的一行用一个缩进了一个制表符的0表示其有空的左子树；若右子树为空，则右子树无须表示。若右子树不为空，则表示完左子树后再表示右子树。

给定一个文本二叉树，求其前序、中序、后序遍历序列。

例题：文本二叉树

样例输入：

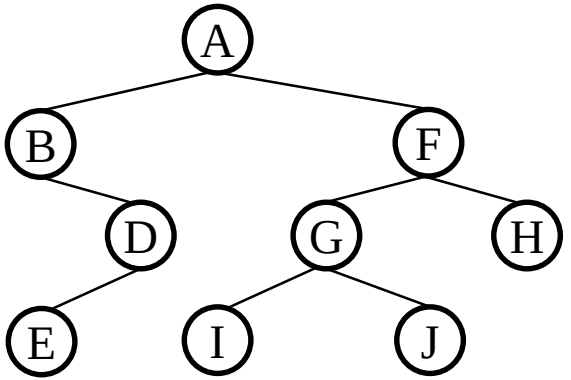
```
A
        B
                0
                D
                        E
        F
                G
                        I
                        J
                H
```



例题：文本二叉树

样例输出：

ABDEFGIJH
BEDAIGJFH
EDBIJGHFA



例题：文本二叉树

```
class BinaryTree:
    ..... 与前同，略
def buildTree(nodes):
    nodesPtr = 0  #正要看nodes里的第几个元素
    def build(level):
        #读取nodesPtr指向的那一个元素，并建立以其为根的子树，该根的层次是level。
        nonlocal nodesPtr
        tree = BinaryTree(nodes[nodesPtr][1]) #建根结点
        nodesPtr += 1 #看下一个元素
        if nodesPtr < len(nodes) and nodes[nodesPtr][0] == level + 1:
            if nodes[nodesPtr][1] != '0':
                tree.addLeft(build(level + 1))
            else:  #没有左子树
                nodesPtr += 1
```

例题：文本二叉树

```
        if nodesPtr < len(nodes) and nodes[nodesPtr][0] == level + 1:
            tree.addRight(build(level + 1))
        return tree
    return build(0)

nodes = [] #nodes元素为 （缩进，数据），例如：[(0, 'A'), (1, 'B'), ......]
               #每个元素代表一个结点，缩进即结点的层次
while True:
    try:
        s = input().rstrip()
        nodes.append((len(s)-1,s.strip()))
    except:
        break
```

例题：文本二叉树

```
tree = buildTree(nodes)
tree.preorderTraversal(lambda x:print(x.data,end=""))
print()
tree.inorderTraversal(lambda x:print(x.data,end=""))
print()
tree.postorderTraversal(lambda x:print(x.data,end=""))
```

*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 fib (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 preorderTravel(self):   #先序遍历非递归生成器函数写法
        stack = [self]
        while len(stack) > 0:
            node = stack.pop()
            if node == None:
                continue
            yield node
            stack.append(node.right)
            stack.append(node.left)  #后入栈的先访问
```

用法：

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

二叉树的非递归遍历

用法：

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

```
def inorderTravel(self,op):
    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]是子树根结点
            stack.pop()
            continue
        if node[1] == 0: #左子树还没有遍历过
            stack.append([node[0].left,0])
            node[1] = 1 #表示左子树遍历过
        elif node[1] == 1: #左子树已经遍历过
            op(node[0])
            stack.pop()
            stack.append([node[0].right, 0])
```

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

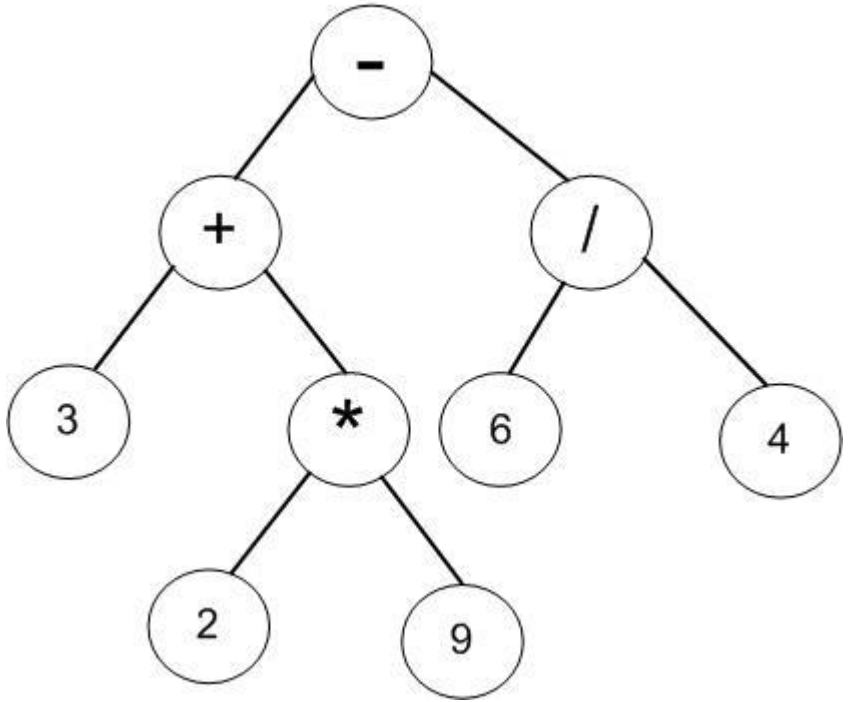
后序遍历得到后缀表达式：

3 2 9 * + 6 4 / -

中序遍历得到运算符中置表达式：

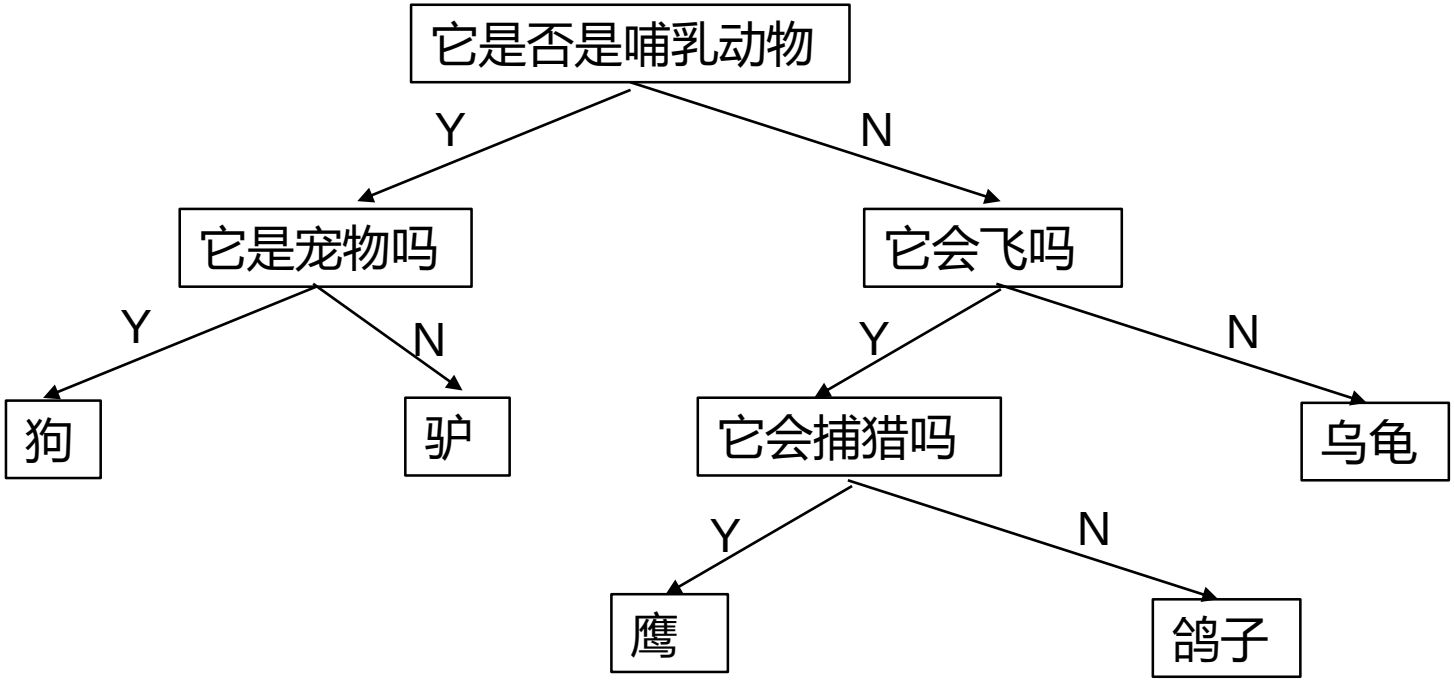
3 + 2 * 9 – 6 / 4

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。



哈夫曼树
(最优二叉树)

木兰围场泰丰湖

最优二叉树

给定n个结点，结点i有权值W_i。要求构造一棵二叉树，叶子结点为给定的结点，且

$$\text{WPL} = \sum_{i=1}^{n} W_i \times L_i$$

最小。Li 是结点i到树根的路径的长度。WPL: Weighted Path Length of Tree

最优二叉树也叫哈夫曼树

最优二叉树的构造

1）开始n个结点位于集合S

2）从S中取走两个权值最小的结点n1和n2，构造一棵二叉树，树根为结点r，r的两个子结点是n1和n2，且$W_r=W_{n1}+W_{n2}$，并将r加入S

3）重复2），直到S中只有一个结点，最优二叉树就构造完毕，根就是S中的唯一结点

证明较麻烦，略

显然，最优二叉树不唯一

哈夫曼编码树

需要对信息中用到的每个字符进行编码。

定长编码方案：每个字符编码的比特数都相同。比如ASCII编码方案。

A 000　C 010　E 100　G 110
B 001　D 011　F 101　H 111

BACADAEAFABBAAAGAH

被编码为以下54个bits:

001000010000011000100000101000001001000000000110000111

哈夫曼编码树

熵编码方案：使用频率高的字符，给予较短编码，使用频率低的字符，给予较长编码，如哈夫曼编码。

A 0　　C 1010　　E 1100　　G 1110
B 100　　D 1011　　F 1101　　H 1111

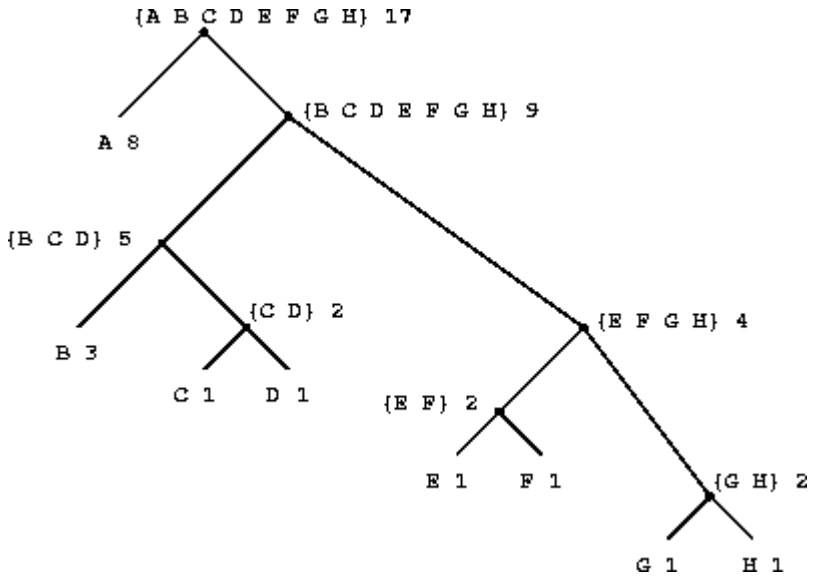
BACADAEAFABBAAAGAH

被编码为以下42个bits:

100010100101101100011010100100000111001111

哈夫曼编码树

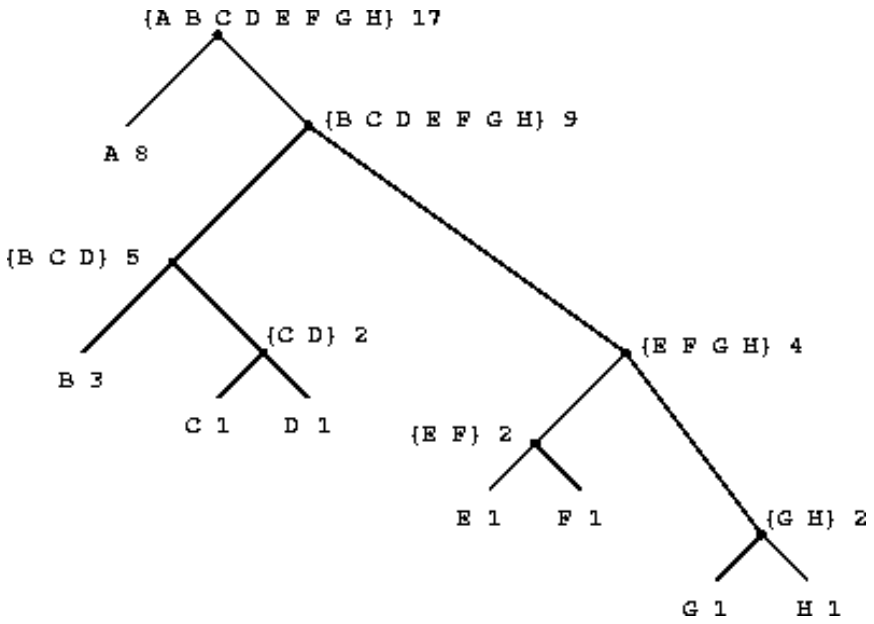
使用可变长编码，需要解决的问题是：如何区分一个编码是一个字符的完整编码，还是另一个字符的编码的前缀。解决办法之一就是采用前缀编码：任何一个字符的编码，都不会是其他字符编码的前缀。



哈夫曼编码树：

- 二叉树
- 叶子代表字符，且每个叶子结点有个权值，权值即该字符的出现频率
- 非叶子结点里存放着以它为根的子树中的所有字符，以及这些字符的权值之和
- 权值仅用来建树，对于字符串的解码和编码没有用处

哈夫曼编码树

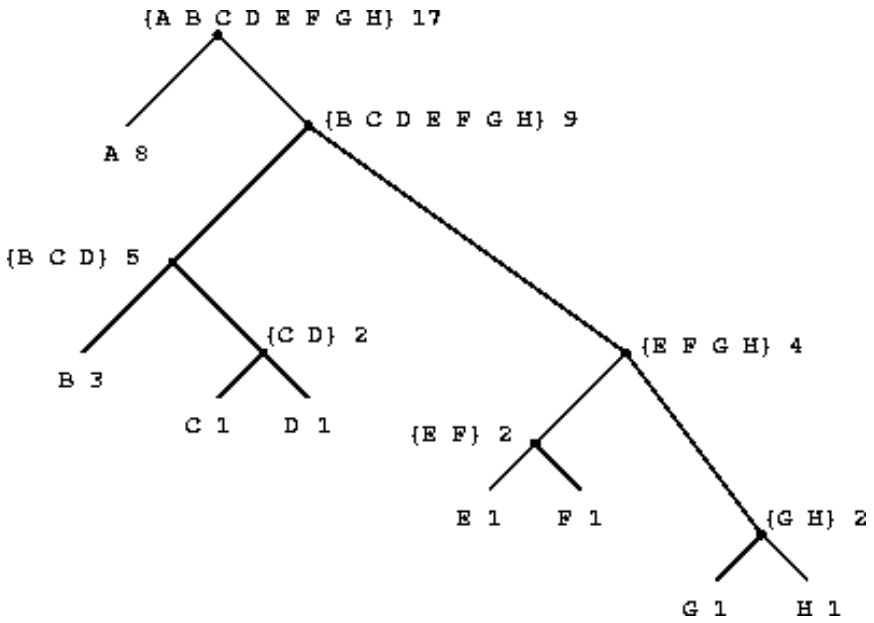


字符的编码过程：

从树根开始，每次往包含该字符的子树走。往左子树走，则编码加上比特1，往右子树走，则编码加上比特0

A 0
B 100
C 1010
G 1110
H 1111

哈夫曼编码树



字符串编码的解码过程：

从树根开始，在字符串编码中碰到一个0，就往左子树走，碰到1，就往右子树走。走到叶子，即解码出一个字符。然后回到树根重复前面的过程。

10001010
BAC

哈夫曼编码树的构造

基本思想：使用频率越高的字符，离树根越近。构造过程和最优二叉树一样

过程：

1．开始时，若有n个字符，则就有n个结点。每个结点的权值就是字符的频率，每个结点的字符集就是一个字符。

2．取出权值最小的两个结点，合并为一棵子树。子树的树根的权值为两个结点的权值之和，字符集为两个结点字符集之并。在结点集合中删除取出的两个结点，加入新生成的树根。

3．如果结点集合中只有一个结点，则建树结束。否则，goto 2

哈夫曼编码树的构造

```
Initial leaves
{(A 8) (B 3) (C 1) (D 1) (E 1) (F 1) (G 1) (H 1)}Merge
{(A 8) (B 3) ({C D} 2) (E 1) (F 1) (G 1) (H 1)}Merge
{(A 8) (B 3) ({C D} 2) ({E F} 2) (G 1) (H 1)}Merge
{(A 8) (B 3) ({C D} 2) ({E F} 2) ({G H} 2)}Merge
{(A 8) (B 3) ({C D} 2) ({E F G H} 4)}Merge
{(A 8) ({B C D} 5) ({E F G H} 4)} Merge
{(A 8) ({B C D E F G H} 9)}Final merge
{({A B C D E F G H} 17)}
```

哈夫曼编码树不唯一

如何快速地在结点集合取出权值最小的两个结点？不要O(n)的笨办法。用"堆"，可以做到O(log(n))

最优二叉树例题

黄石大峡谷

Fence Repair

一块长木板，要切割成长度为$L_1,L_2...L_n$的n块板子。每切一刀的费用，等于被切的那块板子的长度。求最少费用。

Fence Repair

思路：

考虑等价的切割的逆过程，即用n块板子去粘接成最终的长板子。每粘接一次的费用等于粘成的木板长度。

将粘接过程中产生的每个木板，包括最终长木板，都看作一个结点。则粘接的过程可以描述成一棵树的建立过程。将n1,n2粘接成R ， 就相当于建一棵以R为根， n1,n2为子结点的二叉树。最终长板就是最终二叉树的树根。

建树完成后，设n_i到根的路径长度为L_i，则其参加了次L_i粘接,贡献了费用$L_i \times W_i$。要使总费用最小就是 WPL = $\sum_{i=1}^{n} W_i \times Li$ 最小，即最优二叉树问题。