**二叉树的遍历**

1. 树的建立：

Python语言的列表特性使得利用列表来建立树是一种很直白又很简单的方法，但是这种方法使得代码非常繁琐，大量的中括号使得代码的可读性很差，并且在此基础上建立的树很难进行插入等操作，所以在这里我们使用菜单的方式用类来建立树。

1. 算法：

使用类来建立树，先建立一个根实例作为空的根节点，然后定义新的方法来插入左子树和右子树，这里以左子树为例：要注意的是树可能是空的，在树是空的的情况下，直接将节点插入左子树下即可。当树非空时，将原节点下移一位，并将新节点插入二者之间。

1. 代码：

class Binaytree:

#def the root object and an empty tree

def \_\_init\_\_(self, rootObj):

self.root = rootObj

self.lchild = None

self.rchild = None

#insert the left childtree of root Node

def insertLchildtree(self, newNode):

#two different situations, the tree is empty or not

if self.lchild == None:

self.lchild = Binaytree(newNode)

else:

l = self.lchild

self.lchild = Binaytree(newNode)

self.lchild = l

#so the same as the right tree

def insertRchildtree(self, newNode):

if self.rchild == None:

self.rchild = Binaytree(newNode)

else:

r = self.rchild

self.rchild = newNode

self.rchild = r

1. 树的遍历：

我们可以在树的定义类中定义性的方法来实现树的遍历

前序遍历法即先输出根节点，然后分别输出左子树与右子树

中序要求先输出左子树

后续要求先输出右子树

在这里我使用的方法不够智能，只能一步一步地实现，最终需要手动调用方法来实现全二叉树的遍历。

1. 代码实现（全）：

# -\*- coding: utf-8 -\*-

"""

Created on Mon Jun 22 11:19:17 2020

@author: 13988

"""

class Binaytree:

#def the root object and an empty tree

def \_\_init\_\_(self, rootObj):

self.root = rootObj

self.lchild = None

self.rchild = None

#insert the left childtree of root Node

def insertLchildtree(self, newNode):

#two different situations, the tree is empty or not

if self.lchild == None:

self.lchild = Binaytree(newNode)

else:

l = self.lchild

self.lchild = Binaytree(newNode)

self.lchild = l

#so the same as the right tree

def insertRchildtree(self, newNode):

if self.rchild == None:

self.rchild = Binaytree(newNode)

else:

r = self.rchild

self.rchild = newNode

self.rchild = r

#range methord

#we only need to know the different appear in printmode

#we need to know the tree is empty or not

def frontsize(self,root):

if root == None:

return 'the tree is empty'

#we get the root first, then we need to know the left then,the right

print (root.newNode, end = '')

self.frontsize(root.lchild)

self.frontsize(root.rchild)

def middlesize(self, root):

if root == None:

return

self.middlesize(root.lchild)

print(root.newNode, end = '')

self.middlesize(root.rchild)

def latersize(self, root):

if root == None:

return

self.latersize(root.lchild)

self.latersize(root.rchild)

print(root.newNode, end = '')

'''

#they are some new function to get the value of chlid tree and root node

def getlchild(self):

return self.lchild

def getrchlid(self):

return self.rchild

def setrootnode(self, obj):

self.key = obj

def getrootnode(self):

return self.key

'''

a = Binaytree(1)

b = a.insertLchildtree(4)

c = a.insertRchildtree(6)

d = a.insertLchildtree(2)

e = a.insertRchildtree(3)

m = Binaytree(d)

n = m.insertRchildtree(5)