6.4 二叉树的层次遍历

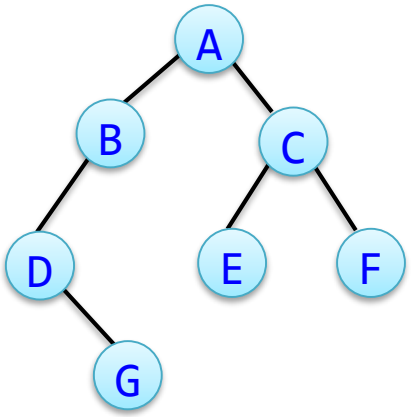
6.4.1 层次遍历过程

若二叉树非空（假设其高度为*h*），则层次遍历的过程如下：

① 访问根结点（第1层）。

② 从左到右访问第2层的所有结点。

③ 从左到右访问第3层的所有结点、…、第*h*层的所有结点。



层次遍历序列为ABCDEFG

6.4.2 层次遍历算法设计

- 在二叉树层次遍历中，对一层的结点访问完后，再按照它们的访问次序对各个结点的左、右孩子顺序访问，这样一层一层进行，先访问的结点其左、右孩子也要先访问，这样与队列的先进先出特点吻合。因此层次遍历算法采用一个队列**qu**来实现。
- 思路：先将根结点***b***进队，在队不空时循环：从队列中出队一个结点***p***，访问它；若它有左孩子结点，将左孩子结点进队；若它有右孩子结点，将右孩子结点进队。如此操作直到队空为止。

```
from collections import deque      #引用双端队列deque
def LevelOrder(bt):                #层次遍历的算法
    qu=deque()                     #将双端队列作为普通队列qu
    qu.append(bt.b)                #根结点进队

    while len(qu)>0:               #队不空循环
        p=qu.popleft()             #出队一个结点
        print(p.data,end=' ')      #访问p结点

        if p.lchild!=None:         #有左孩子时将其进队
            qu.append(p.lchild)

        if p.rchild!=None:         #有右孩子时将其进队
            qu.append(p.rchild)
```

6.4.3 层次遍历算法的应用

【例6.16】采用层次遍历方法设计算法，求二叉树中第k（$1 \leqslant k \leqslant$二叉树高度）层的结点个数。

解法1（自学）

用cnt变量计第*k*层结点个数（初始为0）。设计队列中元素类型为QNode类，包含表示当前结点层次lno和结点引用node两个成员变量。先将根结点进队（根结点的层次为1）。在层次遍历中出队一个结点*p*：

（1）若结点*p*层次大于*k*，返回cnt（继续层次遍历不可能再找到第*k*层的结点）。

（2）若结点*p*是第*k*层的结点（p.lno=*k*），cnt增1。

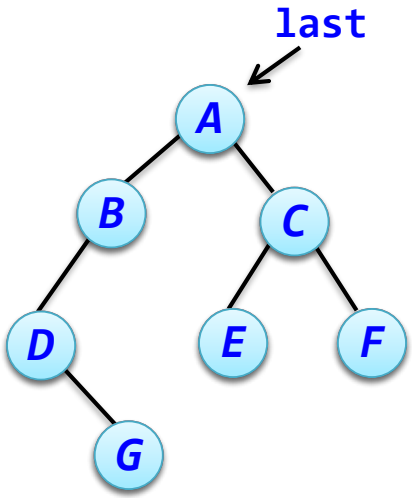
（3）若结点*p*的层次小于*k*，将其孩子结点进队，孩子结点的层次为双亲结点的层次加1。

最后返回cnt。

```
from collections import deque                #引用双端队列deque
class QNode:                                 #队列元素类
   def __init__(self,l,p):                   #构造方法
      self.lev=l                             #结点的层次
      self.node=p                            #结点引用

def KCount1(bt,k):                           #解法1：求二叉树第k层结点个数
   cnt=0                                     #累计第k层结点个数
   qu=deque()                                #定义一个队列qu
   qu.append(QNode(1,bt.b))                  #根结点(层次为1)进队
   while len(qu)>0:                          #队不空循环
      p=qu.popleft()                         #出队一个结点
      if p.lev>k:                            #当前结点的层次大于k，返回cnt
         return cnt
      if p.lev==k:
         cnt+=1                              #当前结点是第k层的结点,cnt增1
      else:                                  #当前结点的层次小于k
         if p.node.lchild!=None:             #有左孩子时将其进队
            qu.append(QNode(p.lev+1,p.node.lchild))
         if p.node.rchild!=None:             #有右孩子时将其进队
            qu.append(QNode(p.lev+1,p.node.rchild))
    return cnt
```

解法2（自学） 层次遍历中某层的最右结点last



last的作用确定一层是否遍历完成！

用cnt变量计第*k*层结点个数（初始为0）。设计队列仅保存结点引用，置当前层次curl=1，用last变量指示当前层次的最右结点（根结点）进队。将根结点进队，队不空循环：

（1）若curl>*k*，返回cnt（继续层次遍历不可能再找到第*k*层的结点）。

（2）否则出队结点*p*，若curl=*k*，表示结点*p*是第*k*层的结点，cnt增1。

（3）若结点*p*有左孩子*q*，将结点*q*进队，有右孩子*q*，将结点*q*进队（总是用*q*表示进队的结点）。

（4）若结点*p*是当前层的最右结点（*p*=last），说明当前层处理完毕，而此时的*q*就是下一层的最右结点，置last=q，curl++进入下一层处理。

```
from collections import deque      #引用双端队列deque
def KCount2(bt,k):                 #解法2：求二叉树第k层结点个数
    cnt=0                          #累计第k层结点个数
    qu=deque()                     #定义一个队列qu
    curl=1                         #当前层次,从1开始
    last=bt.b                      #第1层最右结点
    qu.append(bt.b)                #根结点进队
    while len(qu)>0:               #队不空循环
      if curl>k:                   #当层号大于k时返回cnt,不再继续
         return cnt
      p=qu.popleft()               #出队一个结点
      if curl==k:
         cnt+=1                    #当前结点是第k层的结点,cnt增1
      if p.lchild!=None:           #有左孩子时将其进队
         q=p.lchild
         qu.append(q)
      if p.rchild!=None:           #有右孩子时将其进队
         q=p.rchild
         qu.append(q)
     if p==last:                   #当前层的所有结点处理完毕
        last=q                     #让last指向下一层的最右结点
        curl+=1
    return cnt
```

解法3

层次遍历是从第一层开始，访问一层的全部结点后（此时该层的全部结点已出队）再访问下一层的结点。上一层遍历完毕，队中恰好是下一层的全部结点。若*k*<1，返回0；否则将根结点进队，当前层次curl=1。队不空循环：

（1）若curl=*k*，队中恰好包含该层的全部结点，直接返回队中元素个数（即第*k*层结点个数）。

（2）否则，求出队中元素个数*n*（当前层curl的全部结点个数），循环出队*n*次，每次出队一个结点时将其孩子结点进队。

（3）置curl++，进入下一层处理。

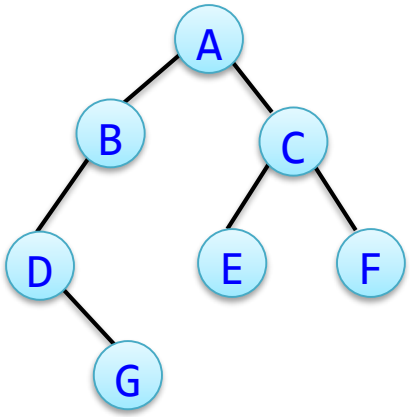
最后返回0（*k*>二叉树高度的情况）。

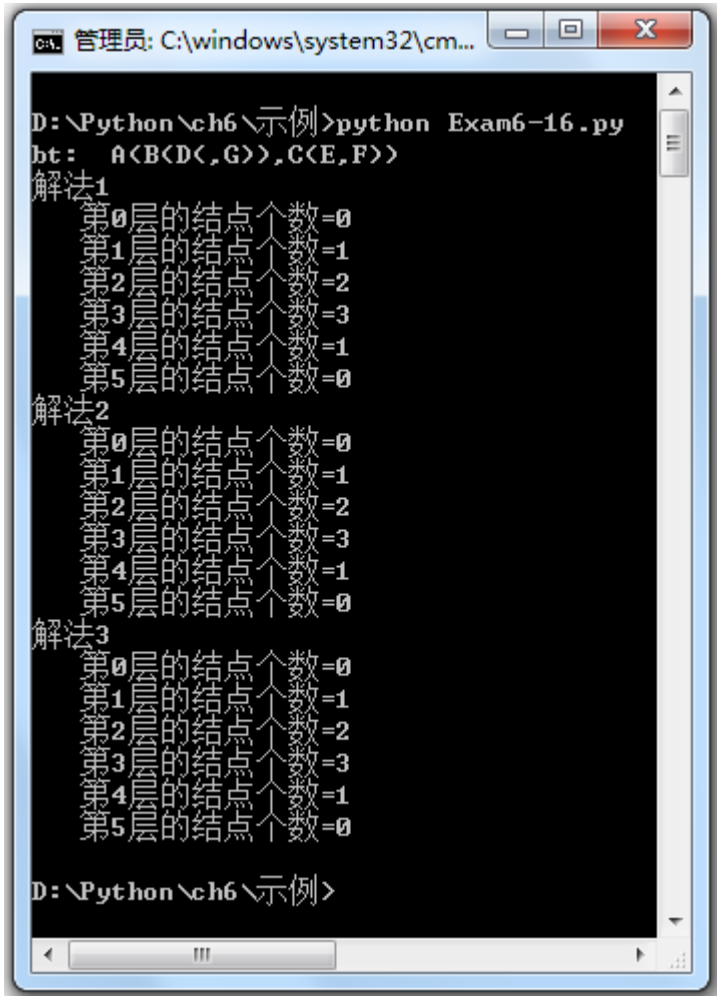
```
from collections import deque                #引用双端队列deque
def KCount3(bt,k):                           #解法3：求二叉树第k层结点个数
    if k<1: return 0                         #k<1返回0
    qu=deque()                               #定义一个队列qu
    curl=1                                   #当前层次,从1开始
    qu.append(bt.b)                          #根结点进队
    while len(qu)>0:                         #队不空循环
      if curl==k:                            #当前层为第k层，返回队中元素个数
         return len(qu)
      n=len(qu)                              #求出当前层结点个数
      for i in range(n):                     #出队当前层的n个结点
         p=qu.popleft()                      #出队一个结点
         if p.lchild!=None:                  #有左孩子时将其进队
            qu.append(p.lchild)
         if p.rchild!=None:                  #有右孩子时将其进队
            qu.append(p.rchild)
      curl+=1                                #转向下一层
    return 0
```

程序验证

```
#主程序
b=BTNode('A')
p1=BTNode('B');p2=BTNode('C')
p3=BTNode('D');p4=BTNode('E')
p5=BTNode('F');p6=BTNode('G')
b.lchild=p1;b.rchild=p2
p1.lchild=p3;p3.rchild=p6
p2.lchild=p4;p2.rchild=p5
bt=BTree()
bt.SetRoot(b)
print("bt:",end='  ');print(bt.DispBTree())
print("解法1")
for i in range(6):
   print("   第%d层的结点个数=%d" %(i,KCount1(bt,i)))
print("解法2")
for i in range(6):
   print("   第%d层的结点个数=%d" %(i,KCount2(bt,i)))
print("解法3")
for i in range(6):
   print("   第%d层的结点个数=%d" %(i,KCount3(bt,i)))
```

程序验证


A
B
C
D
E
F
G


管理员: C:\windows\system32\cm...
D:\Python\ch6\示例>python Exam6-16.py
bt: A(B(D(,G)),C(E,F))
解法1
第0层的结点个数=0
第1层的结点个数=1
第2层的结点个数=2
第3层的结点个数=3
第4层的结点个数=1
第5层的结点个数=0
解法2
第0层的结点个数=0
第1层的结点个数=1
第2层的结点个数=2
第3层的结点个数=3
第4层的结点个数=1
第5层的结点个数=0
解法3
第0层的结点个数=0
第1层的结点个数=1
第2层的结点个数=2
第3层的结点个数=3
第4层的结点个数=1
第5层的结点个数=0
D:\Python\ch6\示例>

【例6.17】采用层次遍历方法设计算法，输出值为*x*的结点的所有祖先。

解：采用和上例解法1类似的思路，设计队列中元素类型为QNode类，包含表示当前结点引用node和双亲pre两个属性。

先将根结点进队（根结点的双亲为None）。在层次遍历中出队一个结点*p*（为队列元素类型而不是二叉树结点类型）：

① 若结点*p*为*x*结点（p.node.data=*x*），从结点*p*出发通过队列元素回推求出所有祖先结点res（类似用队列求解迷宫路径），返回res。

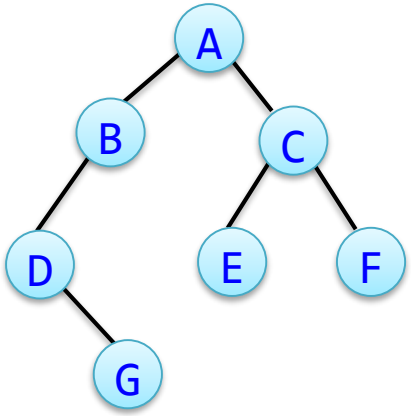
② 否则，将结点*p*的孩子结点进队，注意二叉树中孩子结点p.node.lchild和p.node.rchild的双亲结点均为结点*p*。

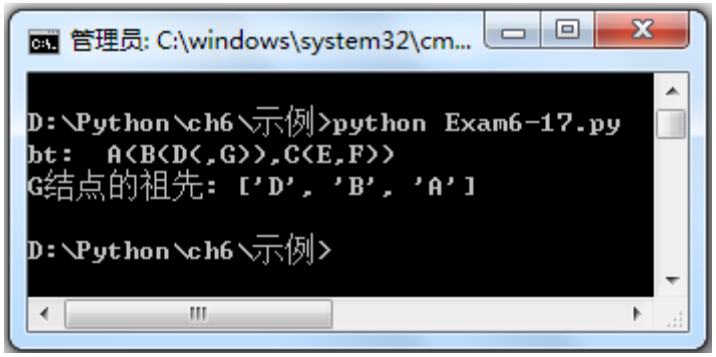
```
from collections import deque                    #引用双端队列deque
class QNode:                                     #队列元素类
    def __init__(self,p,pre):                    #构造方法
        self.node=p                              #当前结点引用
        self.pre=pre                             #当前结点的双亲结点

def Ancestor4(bt,x):                             #层次遍历求x结点的祖先
    res=[]                                       #存放x结点的祖先
    qu=deque()                                   #定义一个队列qu
    qu.append(QNode(bt.b,None))                  #根结点(双亲为None)进队
    while len(qu)>0:                             #队不空循环
      p=qu.popleft()                             #出队一个结点
      if p.node.data==x:                         #当前结点p为x结点
          q=p.pre                                #q为双亲
          while q!=None:                         #找到根结点为止
              res.append(q.node.data)
              q=q.pre
          return res
      if p.node.lchild!=None:                    #有左孩子时将其进队
          qu.append(QNode(p.node.lchild,p))      #置其双亲为p
      if p.node.rchild!=None:                    #有右孩子时将其进队
          qu.append(QNode(p.node.rchild,p))      #置其双亲为p
    return res
```

程序验证

```
#主程序
b=BTNode('A')
p1=BTNode('B');p2=BTNode('C')
p3=BTNode('D');p4=BTNode('E')
p5=BTNode('F');p6=BTNode('G')
b.lchild=p1;b.rchild=p2
p1.lchild=p3;p3.rchild=p6
p2.lchild=p4;p2.rchild=p5
bt=BTree()
bt.SetRoot(b)
print("bt:",end='  ')
print(bt.DispBTree())
x='G'
print(x+"结点的祖先:",Ancestor4(bt,x))
```



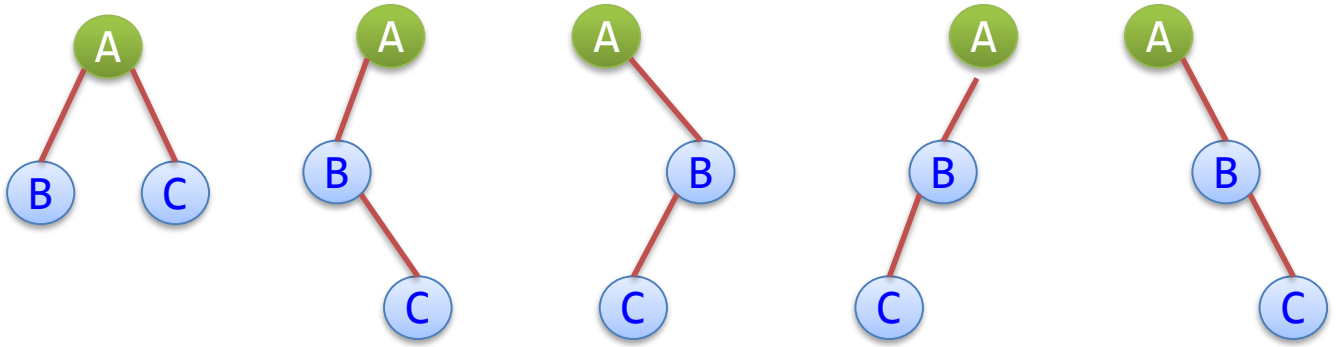


6.5 二叉树的构造

6.5.1 由先序/中序序列或后序/中序序列构造二叉树

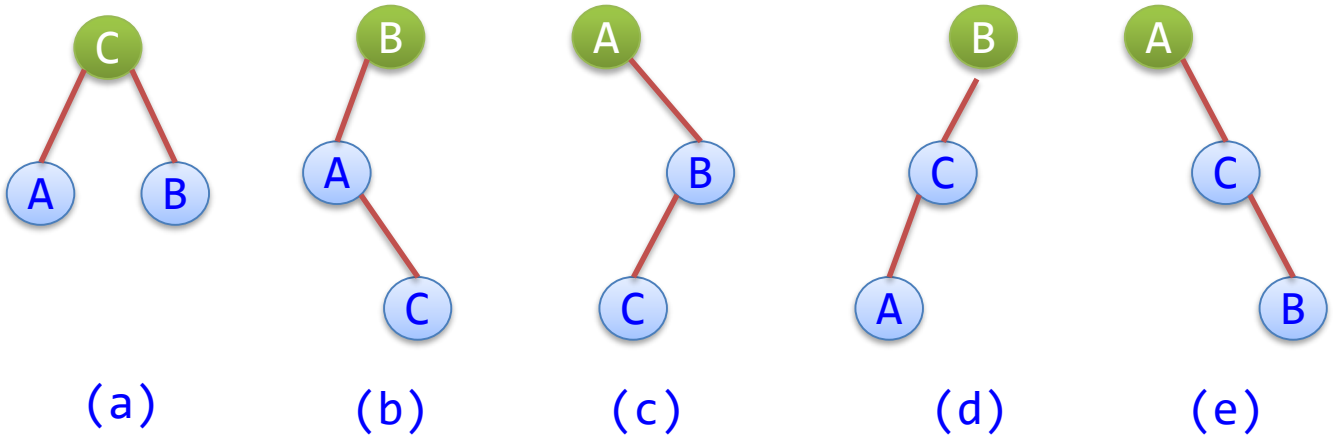
- 一棵所有结点值不同的二叉树，其先序、中序、后序和层次遍历都是唯一的，也就是说一棵这样的二叉树，不可以有两种不同的先序遍历序列，也不可能有两种不同的中序序列。
- 二叉树的构造就是给定某些遍历序列，反过来唯一地确定该二叉树。

二叉树 遍历序列	图(a)的二叉树	图(b)的二叉树	图(c)的二叉树	图(d)的二叉树	图(e)的二叉树
先序遍历序列	***ABC***	***ABC***	***ABC***	***ABC***	***ABC***
中序遍历序列	***BAC***	***BCA***	***ACB***	***CBA***	***ABC***
后序遍历序列	***BCA***	***CBA***	***CBA***	***CBA***	***CBA***



从中看到，对于不同形态的二叉树：

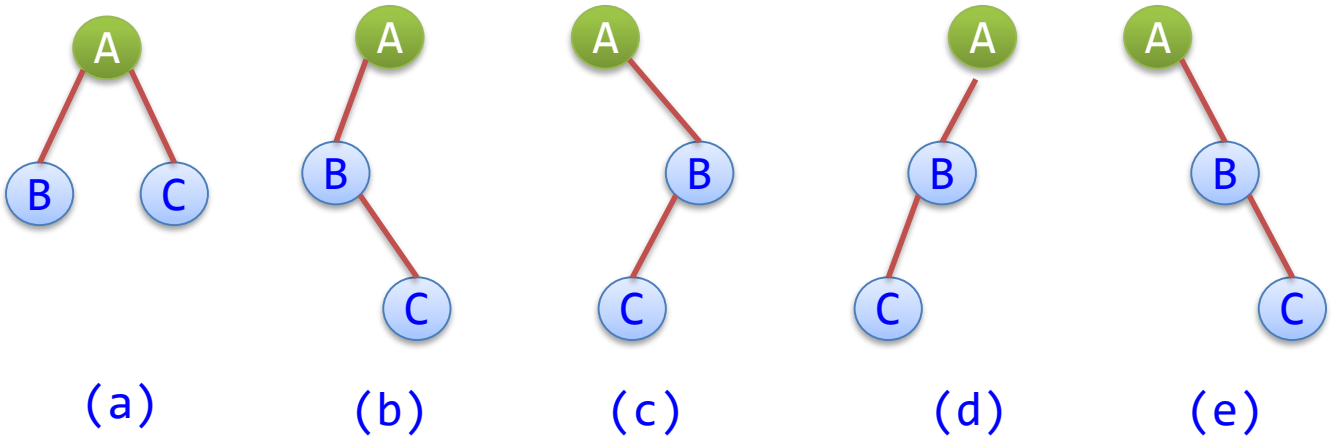
- 中序遍历序列可能相同，如下图的5棵二叉树，它们的中序遍历序列均为ACB。



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

从中看到，对于不同形态的二叉树：

- 先序遍历序列可能相同；
- 中序遍历序列可能相同；
- 后序遍历序列可能相同；
- 先序遍历序列和后序遍历序列可能都相同（图(d)和(e)的先序遍历序列和后序遍历序列均相同）。

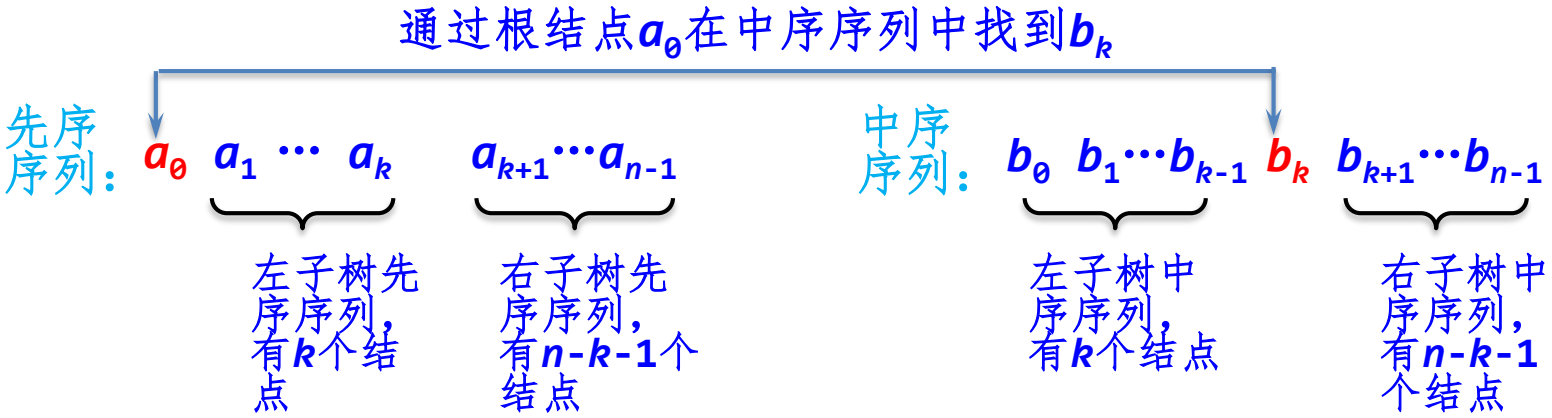


实际上，对于含2个或者以上结点的二叉树，在先序、中序和后序遍历序列中：

- 由先序遍历序列和中序遍历序列能够唯一确定一棵二叉树。
- 由后序遍历序列和中序遍历序列能够唯一确定一棵二叉树。
- 由先序遍历序列和后序遍历序列不能唯一确定一棵二叉树。

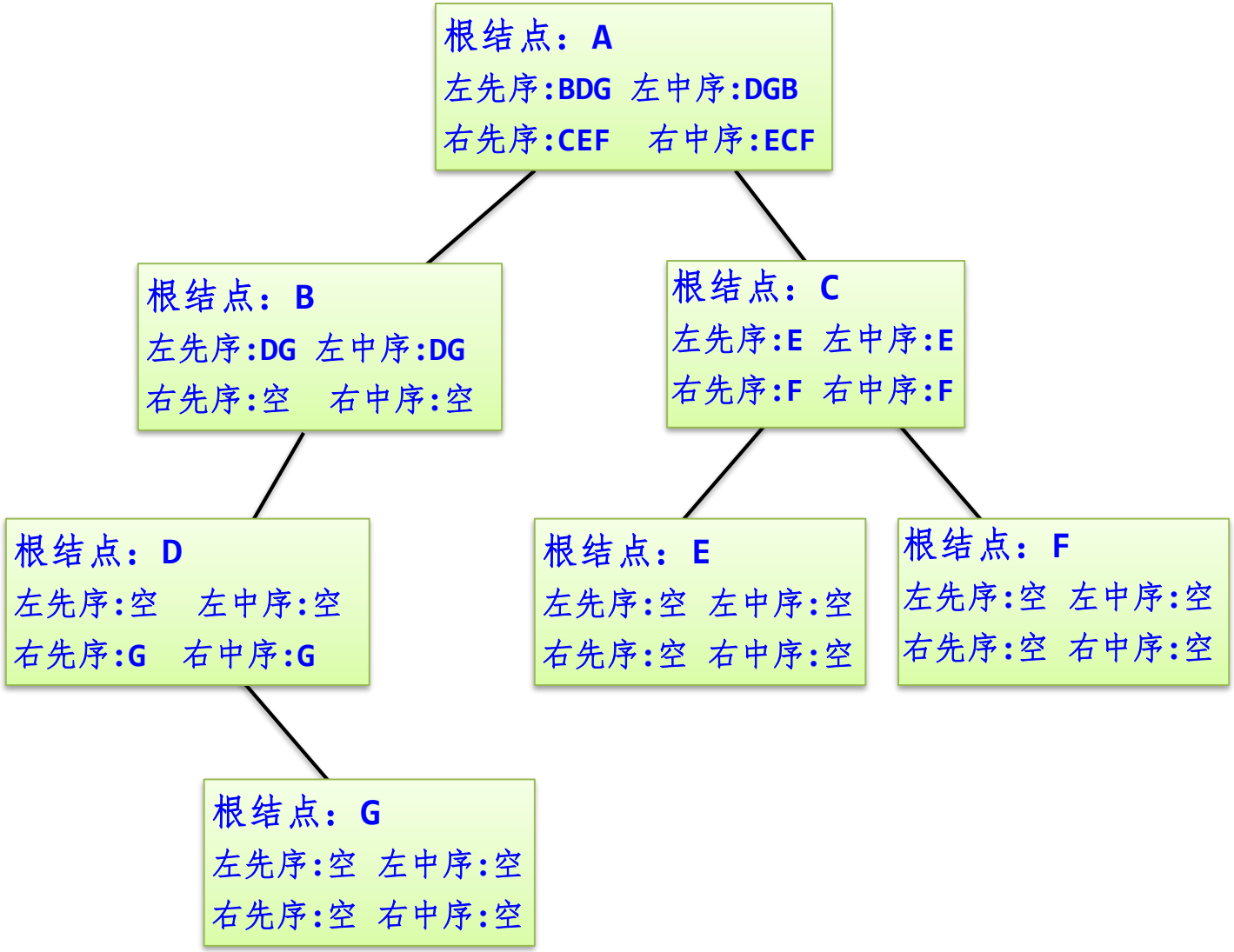
定理6.1 任何n（$n\geq 0$）个不同结点的二叉树，都可由它的中序序列b和先序序列a唯一地确定。

- 由a_0（根结点）找到b_k。



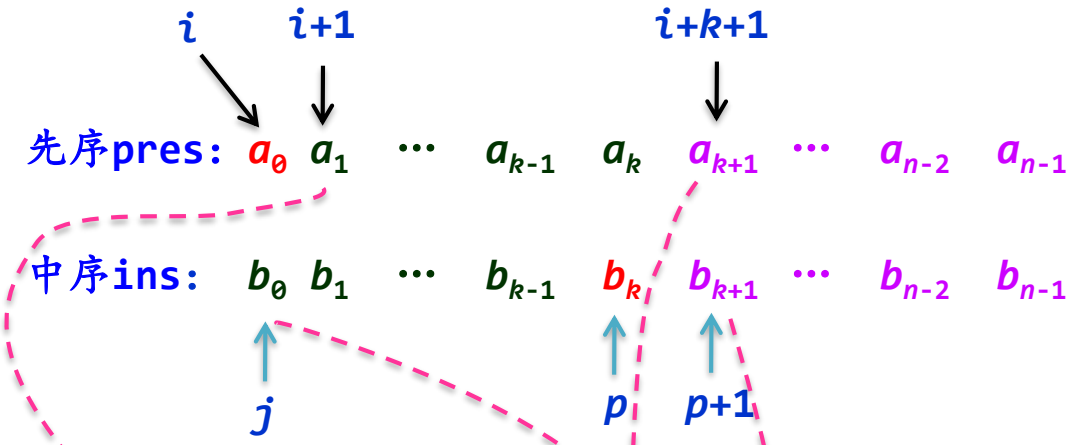
- 若b_k前面有k个结点，则左子树有k个结点，右子树有$n-k-1$个结点。
- 可以求出左右子树的中序序列和先序序列。
- 这样根结点是确定的，左右子树也是确定的，则该二叉树是确定的。

已知先序序列为ABDGCEF，中序序列为DGBAECF，则构造二叉树的过程：



```
def CreateBTree1(pres,ins):       #由先序序列pres和中序序列ins构造二叉链
    bt=BTree()
    bt.b=_CreateBTree1(pres,0,ins,0,len(pres))
    return bt
```

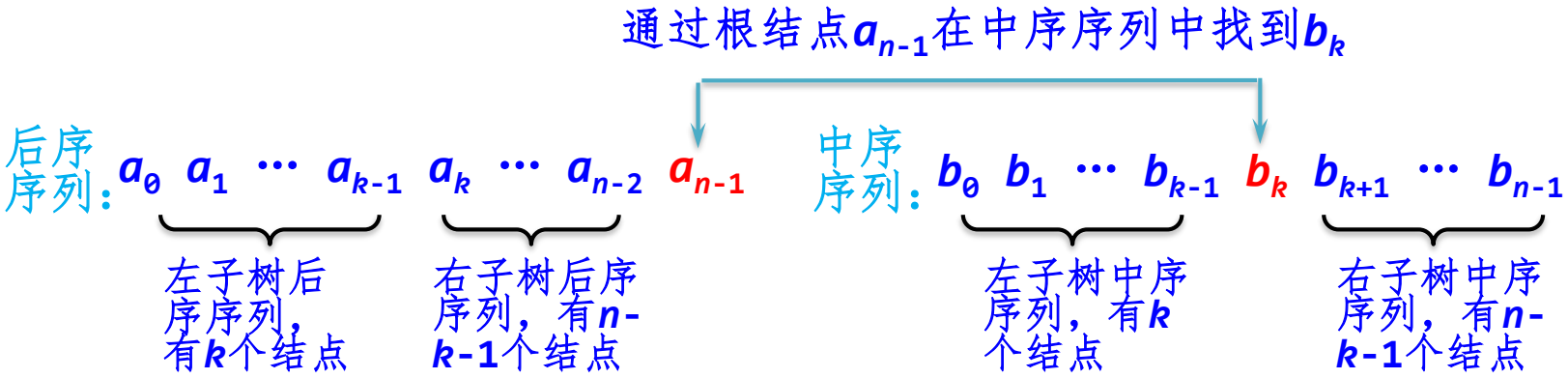
由先序序列pres[i..i+n-1]和中序序列ins[j..j+n-1]创建二叉链t




```
def _CreateBTree1(pres,i,ins,j,n):                    #被CreateBTree1调用
    if n<=0: return None
    d=pres[i]                                         #取根结点值d
    t=BTNode(d)                                       #创建根结点(结点值为d)
    p=ins.index(d)                                    #在ins中找到根结点的索引p
    k=p-j                                             #确定左子树中结点个数k
    t.lchild=_CreateBTree1(pres,i+1,ins,j,k)                  #递归构造左子树
    t.rchild=_CreateBTree1(pres,i+k+1,ins,p+1,n-k-1)          #递归构造右子树
    return t
```

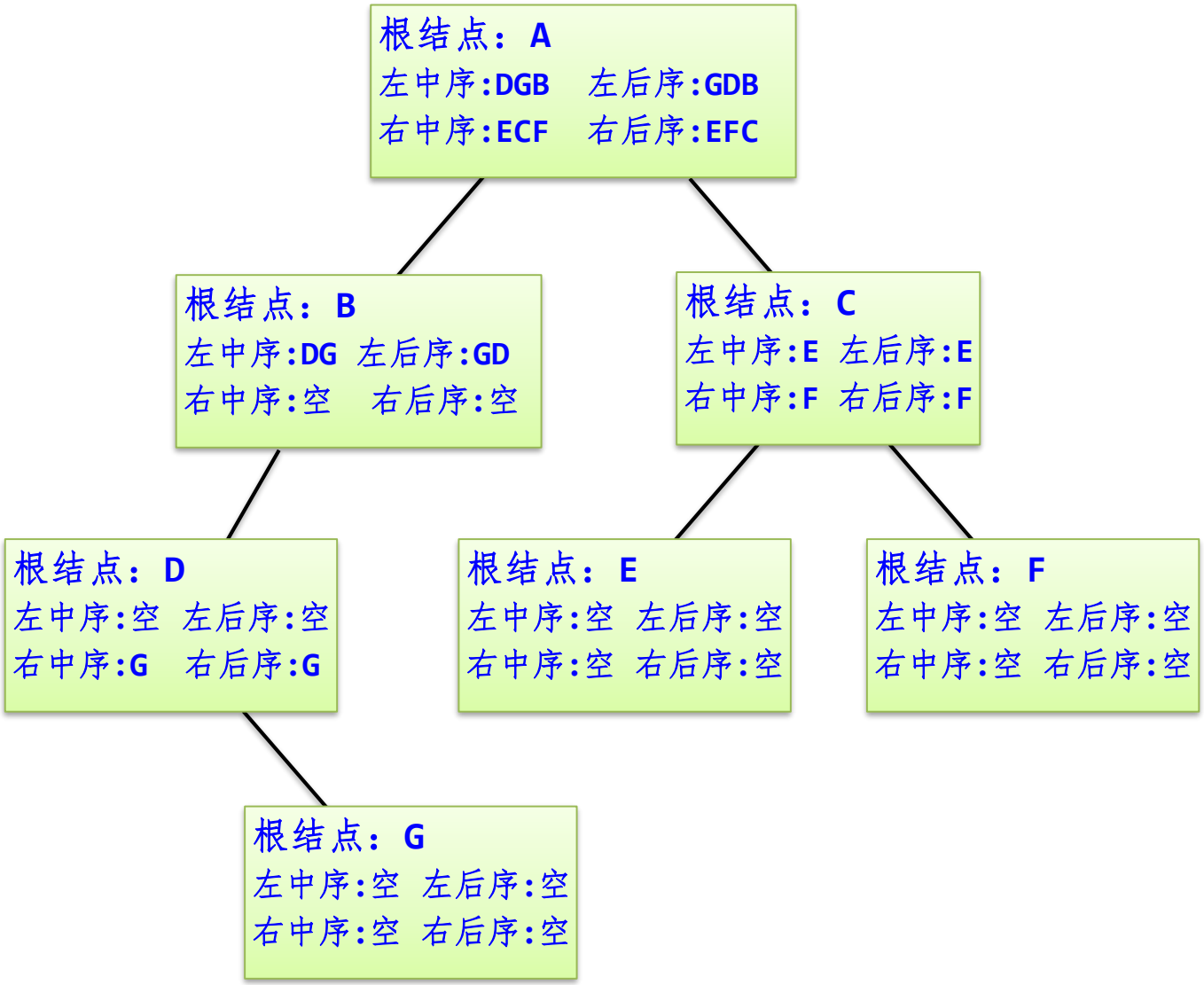
定理6.2 任何n（$n\geq 0$）个不同结点的二叉树，都可由它的中序序列和后序序列唯一地确定。

- 由a_{n-1}（根结点）找到b_k。



- 若b_k前面有k个结点，则左子树有k个结点，右子树有$n-k-1$个结点。
- 可以求出左右子树的中序序列和后序序列。
- 这样根结点是确定的，左右子树也是确定的，则该二叉树是确定的。

已知中序序列为DGBAECF，后序序列为GDBEFCA，则构造二叉树的过程



```
def CreateBTree2(posts,ins):   #由后序序列posts和中序序列ins构造二叉链
    bt=BTree()
    bt.b=_CreateBTree2(posts,0,ins,0,len(posts))
    return bt;
```

由后序序列posts[i..i+n-1]和中序序列ins[j..j+n-1]创建二叉链t

```
def _CreateBTree2(posts,i,ins,j,n):          #被CreateBTree2调用
  if n<=0: return None
  d=posts[i+n-1]                             #取后序序列尾元素即根结点值d
  t=BTNode(d)                                #创建根结点(结点值为d)
  p=ins.index(d)                             #在ins中找到根结点的索引
  k=p-j                                      #确定左子树中结点个数k
  t.lchild=_CreateBTree2(posts,i,ins,j,k)          #递归构造左子树
  t.rchild=_CreateBTree2(posts,i+k,ins,p+1,n-k-1)  #递归构造右子树
  return t
```

【例6.18】若某非空二叉树的先序序列和后序序列正好相同，则该二叉树的形态是什么？

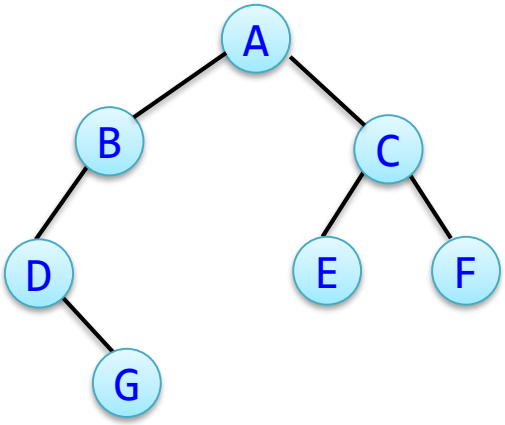
二叉树的先序序列是DLR，后序序列是LRD。

D L R = L R D

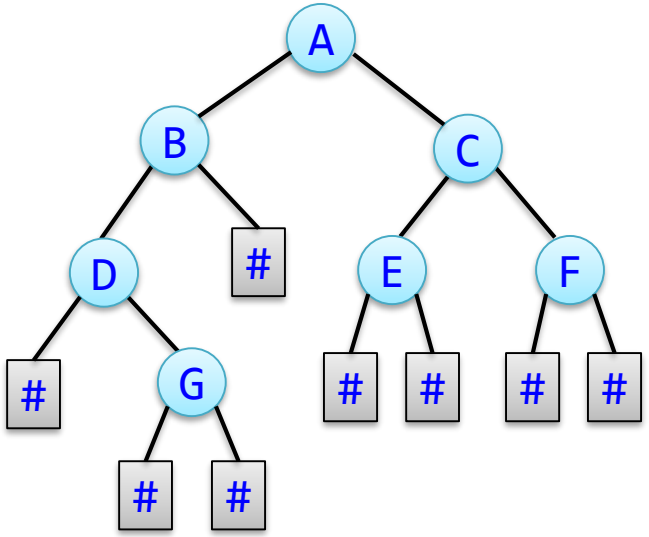
则L和R均为空。所以满足条件的二叉树只有一个根结点。

6.5.2* 序列化和反序列化

仅仅讨论先序遍历序列化和反序列化。



加上外部结点



先序遍历

序列化序列 ⟶ "ABD#G###CE##F##"

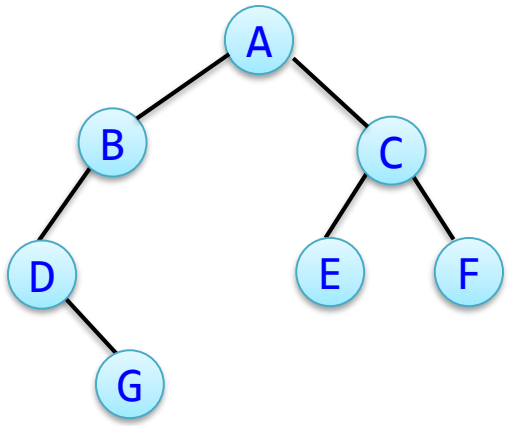
```
def PreOrderSeq(bt):                          #二叉树bt的序列化
    return _PreOrderSeq(bt.b)

def _PreOrderSeq(t):
    if t==None: return ["#"]
    s=[t.data]                                #含根结点
    s+=_PreOrderSeq(t.lchild)                 #产生左子树的序列化序列
    s+=_PreOrderSeq(t.rchild)                 #产生右子树的序列化序列
    return s
```

序列化序列

"ABD#G###CE##F##"

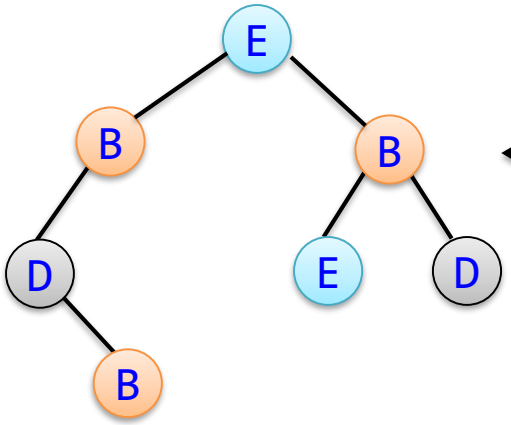
反序列化



```
def CreateBTree3(s):                          #由序列化序列s创建二叉链：反序列化
    bt=BTree()
    it=iter(s)                                #定义s的迭代器lt
    bt.SetRoot(_CreateBTree3(it))
    return bt

def _CreateBTree3(it):
    try:
        d=next(it)                            #取下一个元素d
        if d=="#": return None                #若d为"#"，返回空
        t=BTNode(d)                           #创建根结点(结点值为d)
        t.lchild=_CreateBTree3(it)            #递归构造左子树
        t.rchild=_CreateBTree3(it)            #递归构造右子树
        return t                              #返回根结点
    except StopIteration:
        return None                           #若已经取完，返回空
```

由于反序列化构造二叉树过程中不像先序/中序和后序/中序那样需要根结点值的比较，所以适合构造结点值相同的二叉树。



← 无法由先序序列和中序序列构造！

先序序列：EBDBBED
中序序列：DBBEEBD