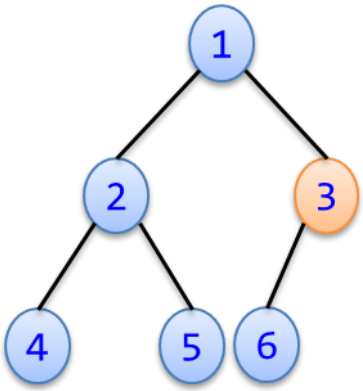
6.2 二叉树



6.2.1 二叉树的概念

1. 二叉树的递归定义

- 二叉树是n个结点的有限集，可分为两种情形：

  1）如果n=0, 则是一棵空二叉树；

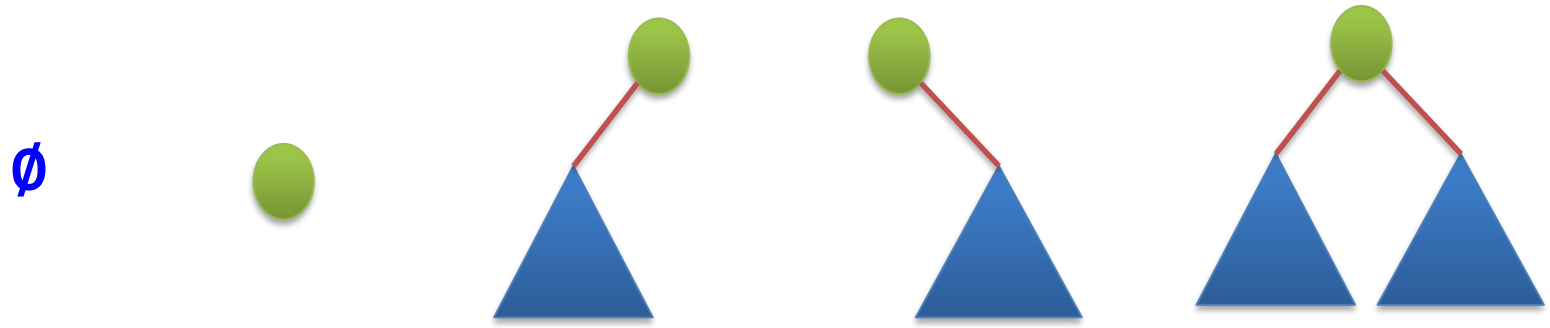
  2）如果n>0, 则它包含一个根节点，而剩余的结点可分为两个不相交的子集，分别构成根节点的左子树和右子树。

提问 二叉树与度为2的树的关系？

提问 二叉树与度为2的树的关系？

- 度为2的树至少有3个结点，而二叉树的结点数可以为0。
- 度为2的树不区分子树的次序，而二叉树中的每个结点最多有两个孩子结点，且必须要区分左右子树，即使在结点只有一棵子树的情况下也要明确指出该子树是左子树还是右子树。

归纳起来，二叉树的5种形态：



(a) 空二叉树

(b) 只有一个根结点的二叉树

(c) 右子树为空的二叉树

(d) 左子树为空的二叉树

(e) 左、右子树非空的二叉树

2.二叉树抽象数据类型的描述

ADT BTree

{

数据对象：

D={a_i | 0≤i≤n-1，n≥0}　//为了简单，除了特别说明外假设结点值为char

数据关系：

R={r}

r={<a_i，a_j> | a_i，a_j∈D，　0≤i，j≤n-1，当n=0时，称为空二叉树；否则其中有一个根结点，其他结点构成根结点的互不相交的左、右子树，该左、右两棵子树也是二叉树 }

基本运算：

SetRoot(b)：设置二叉树的根结点为b。

DispBTree ()：返回二叉树的括号表示串。

FindNode(x)：在二叉树中查找值为x的结点。

int Height()：求二叉树的高度。

…

}

归为三类

查找类　插入类　删除类

6.2.2 二叉树性质

性质1 非空二叉树上叶结点数等于双分支结点数加1。设度为i的结点个数为n_i，则有 $n_0=n_2+1$。

证明：

- 总结点数$n=n_0+n_1+n_2$。
- 除根结点外每一结点都是某一结点的孩子结点，故孩子结点总数为$n-1$；
- 度为i的结点有i个孩子，故孩子总数为n_1+2n_2。
- 即 $n-1=n_1+2n_2$ 。
- 则$n_1+2n_2=n_0+n_1+n_2-1$，求出$n_0=n_2+1$。

归纳

在二叉树中计算结点时常用的关系式有：①所有结点的度之和=$n-1$ ②所有结点的度之和＝n_1+2n_2 ③$n=n_0+n_1+n_2$。

【例6.6】一棵含有882个结点的二叉树中有365个叶子结点，求度为1的结点个数和度为2的结点个数。

解：这里$n=882$，$n_0=365$。

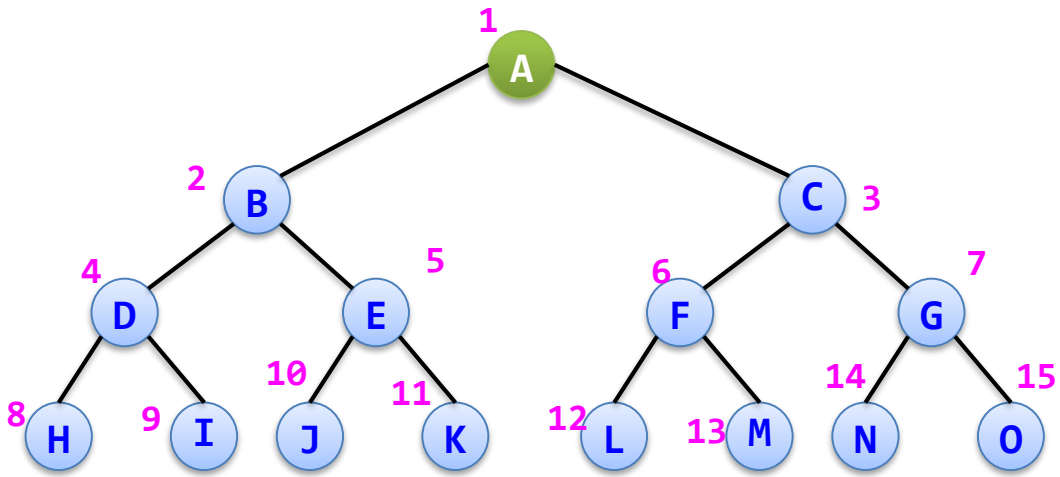
- 由二叉树的性质1可知$n_2=n_0-1=364$。
- $n=n_0+n_1+n_2$，即$n_1=n-n_0-n_2=882-365-364=153$。
- 所以该二叉树中度为1的结点和度为2的结点个数分别是153和364。

性质2 非空二叉树上第i层上至多有2^{i-1}个结点，这里应有$i \geqslant 1$。

性质3 高度为h的二叉树至多有2^h-1个结点（$h \geqslant 1$）。

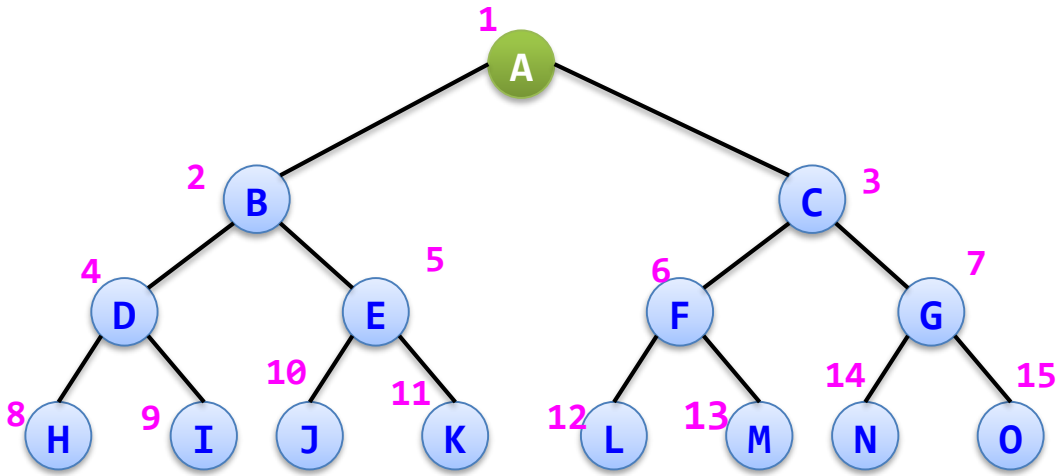
3．满二叉树和完全二叉树

- 在一棵二叉树中，若所有分支结点都有左孩子结点和右孩子结点，并且叶子结点都集中在二叉树的最下一层，则称为满二叉树。
- 对满二叉树中的结点按层序连续编号，约定编号从树根为1开始，按照层数从小到大、同一层从左到右的次序进行。
- 满二叉树也可以从结点个数和树高度之间的关系来定义，即一棵高度为h且有2^h-1个结点的二叉树称为满二叉树。



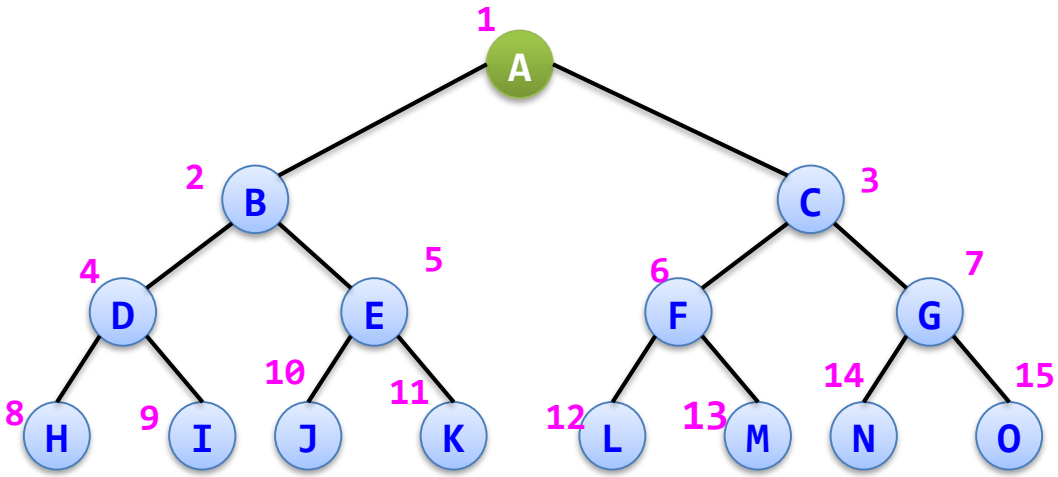
满二叉树的特点如下：

- 叶子结点都在最下一层；
- 只有度为0和度为2的结点；
- 含n个结点的满二叉树的高度为$\log_2(n+1)$。



n=15

$h=\log_2(n+1)=4$



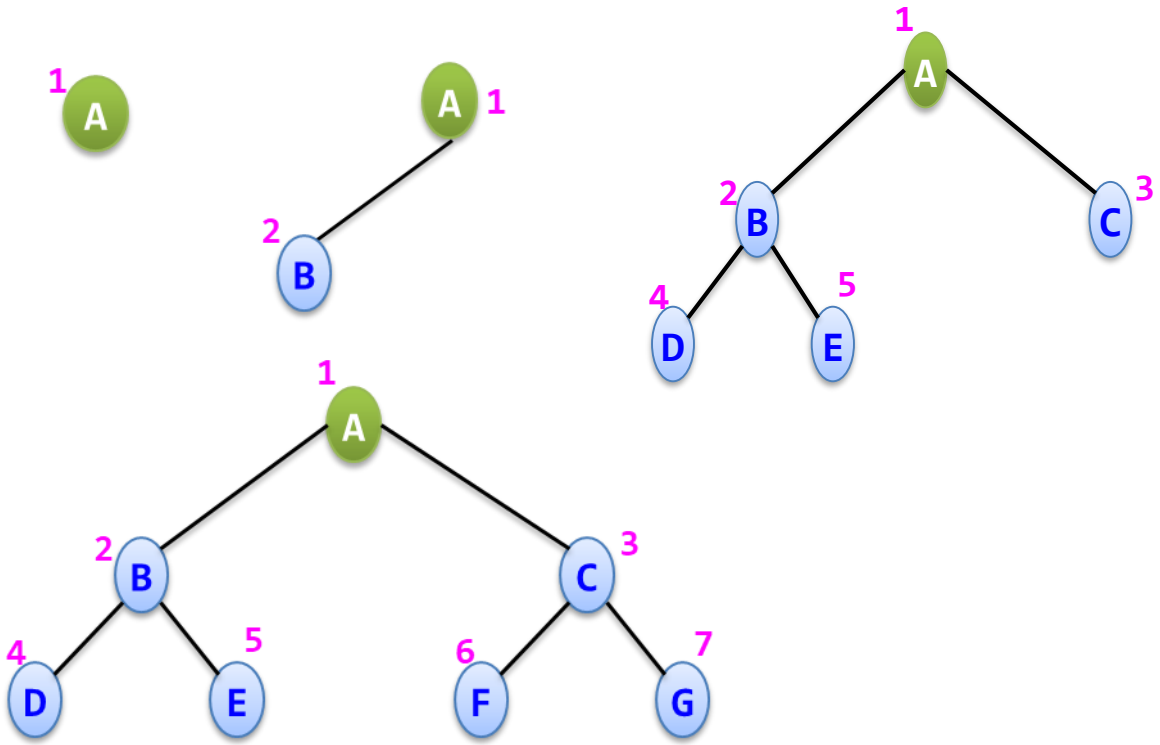
设满二叉树有n个结点，编号为1,2，…,n

左小孩为偶数，右小孩为奇数；

- 结点i的左小孩是2i，2i≤n
- 结点i的右小孩是2i+1, 2i+1≤n
- 结点i的双亲是⌊i/2⌋, 2≤i≤n
- 结点i的层号为$\lfloor \log_2 i \rfloor + 1 = \lceil \log_2 (i+1) \rceil$, 1≤i≤n.

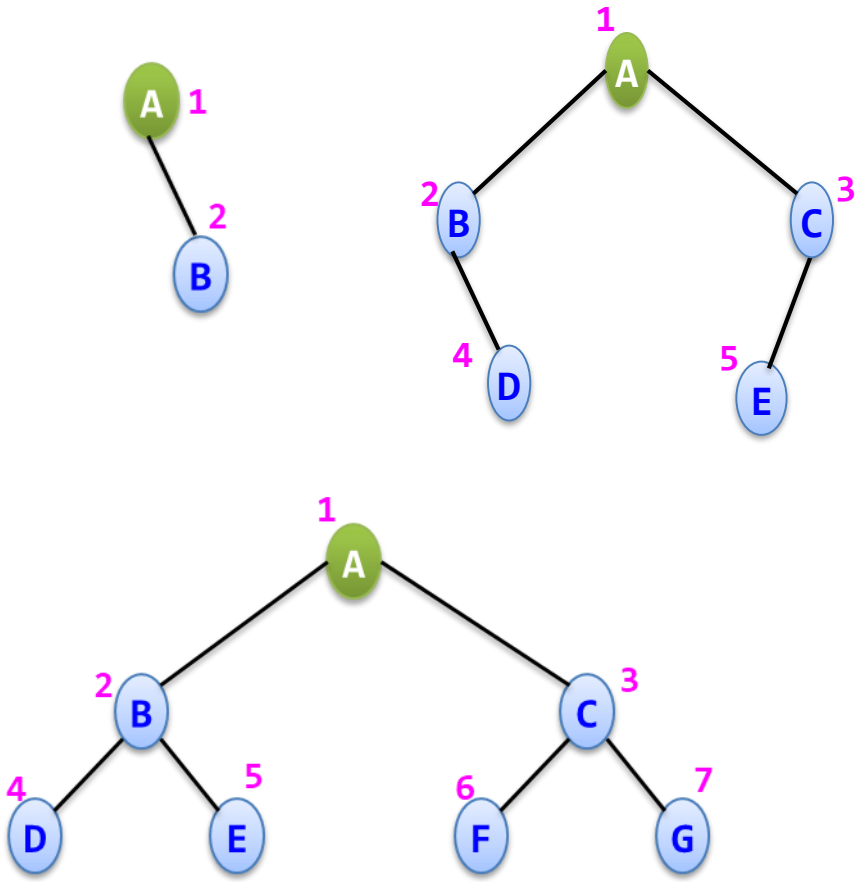
完全二叉树

- 深度为k有n个结点的二叉树，当且仅当每一个结点都与同深度的满二叉树中编号从1到n的结点一一对应，则称之为完全二叉树。



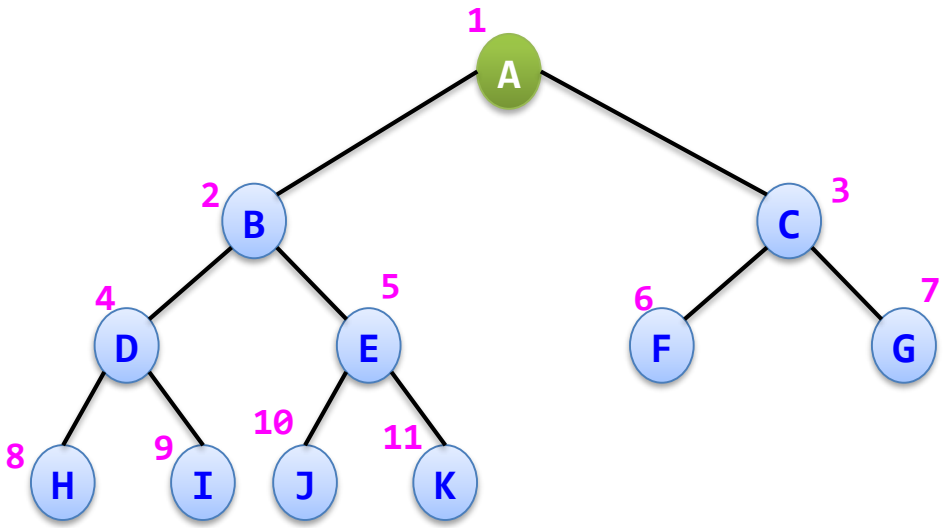
满二叉树一定是完全二叉树，反之不成立。

非完全二叉树实例



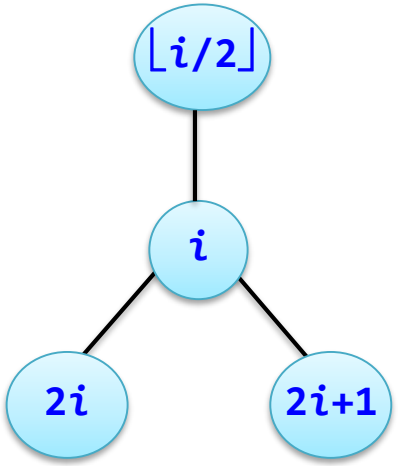
完全二叉树的特点如下：

- 叶子结点只可能出现在最下面两层中；
- 最大层中的叶子结点，依次排在该层最左边的位置上；
- 如果有度为1的结点，只可能有一个，且该结点只有左孩子而无右孩子。

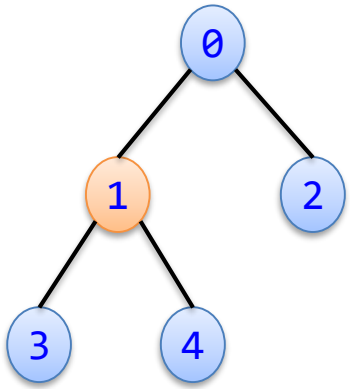


（1）若完全二叉树的根结点编号为1，对于编号为i（$1 \leqslant i \leqslant n$）的结点有：

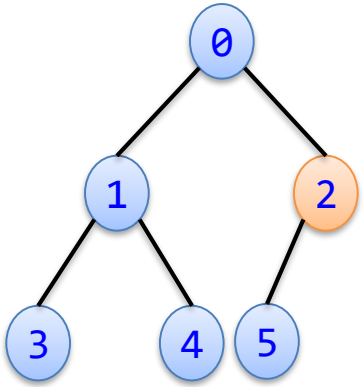
- 若编号为i的结点有左孩子结点，则左孩子结点的编号为$2i$；若编号为i的结点有右孩子结点，则右孩子结点的编号为$2i+1$。
- 若编号为i的结点有左兄弟结点，左兄弟结点的编号为$i-1$，若有右兄弟结点，右兄弟结点的编号为$i+1$。
- 若编号为i的结点有双亲结点，其双亲结点的编号为$\lfloor i/2 \rfloor$。



- 若$i \leqslant \lfloor n/2 \rfloor - 1$，则编号为$i$的结点为分支结点，否则为叶子结点，也就是说，最后一个分支结点的编号为$\lfloor n/2 \rfloor - 1$。
- 若n为奇数，则$n_1=0$，每个分支结点都是双分支结点；若n为偶数，则$n_1=1$，只有一个单分支结点。



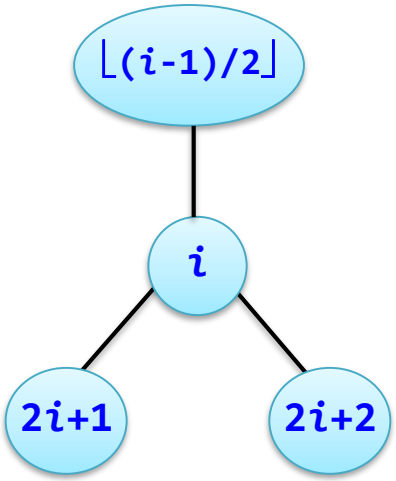
$n=5$ ⇨ $n_1=0$，最后分支结点为$n/2-1=1$



$n=6$ ⇨ $n_1=1$，最后分支结点为$n/2-1=2$

（2）若完全二叉树的根结点编号为0，对于编号为i（0≤i≤n-1）的结点有：

- 若编号为i的结点有左孩子结点，则左孩子结点的编号为2i+1；若编号为i的结点有右孩子结点，则右孩子结点的编号为2i+2。
- 若编号为i的结点有左兄弟结点，左兄弟结点的编号为i-1，若有右兄弟结点，右兄弟结点的编号为i+1。
- 若编号为i的结点有双亲结点，其双亲结点的编号为⌊(i-1)/2⌋。



性质4 具有n个（$n>0$）结点的完全二叉树的高度为$\lceil \log_2(n+1) \rceil$或$\lfloor \log_2 n \rfloor+1$。

由完全二叉树的定义和树的性质3可推出。

归纳

- 一棵完全二叉树中，由结点总数n可以确定其树形。
- n_1只能是0或1，当n为偶数时，$n_1=1$，当n为奇数时，$n_1=0$。
- 编号为i的结点层次恰好为$\lceil \log_2(i+1) \rceil$或者$\lfloor \log_2 i \rfloor+1$。

【例6.8】一棵完全二叉树中有501个叶子结点，则至少有多少个结点。

解：该二叉树中有，$n_0=501$。

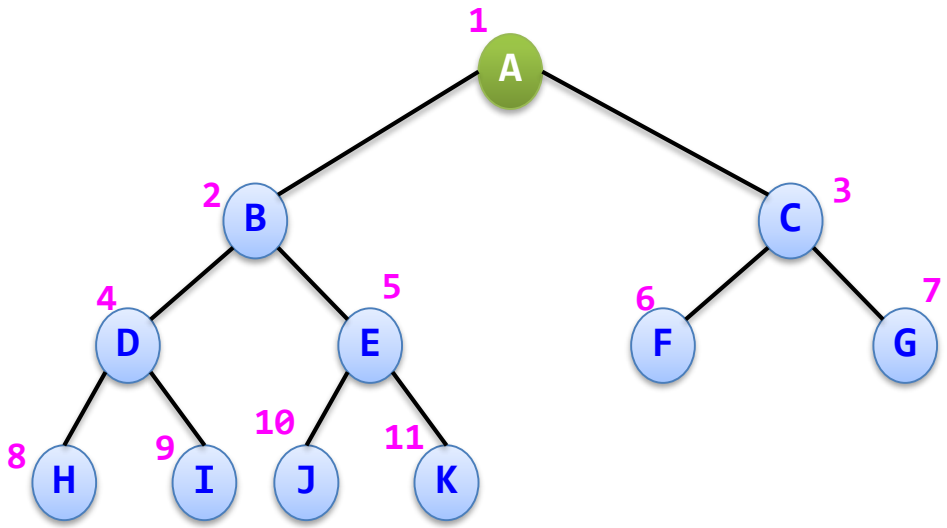
- 由二叉树性质1可知$n_0=n_2+1$，所以$n_2=n_0-1=500$。
- $n=n_0+n_1+n_2=1001+n_1$，由于完全二叉树中$n_1=0$或$n_1=1$，则$n_1=0$时结点个数最少，此时$n=1001$，即至少有1001个结点。

6.2.3 二叉树存储结构

1．二叉树的顺序存储结构

- 对于完全二叉树（或满二叉树），树中结点层序编号可以唯一地反映出结点之间的逻辑关系，故可以用一维数组按从上到下、从左到右的顺序存储树中所有结点值，通过数组元素的下标关系反映完全二叉树或满二叉树中结点之间的逻辑关系。

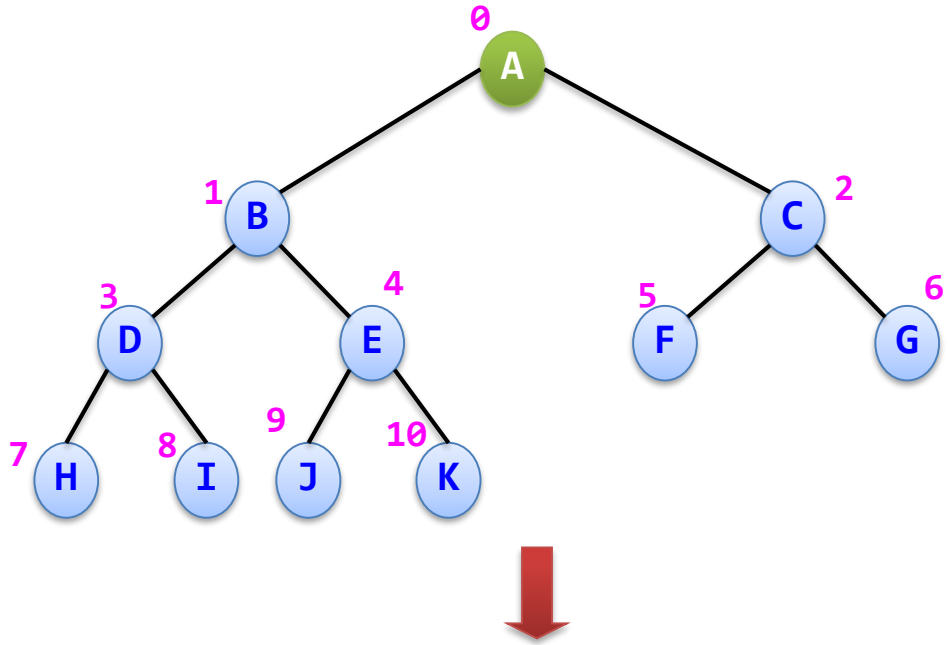
一棵完全二叉树的顺序存储结构



位置	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
sb	A	B	C	D	E	F	G	H	I	J	K	#	#	#	#

sb的下标从1开始

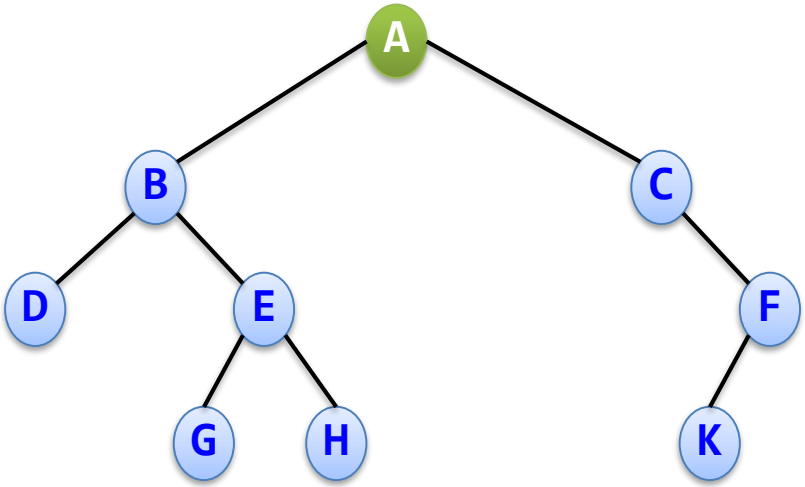
或者



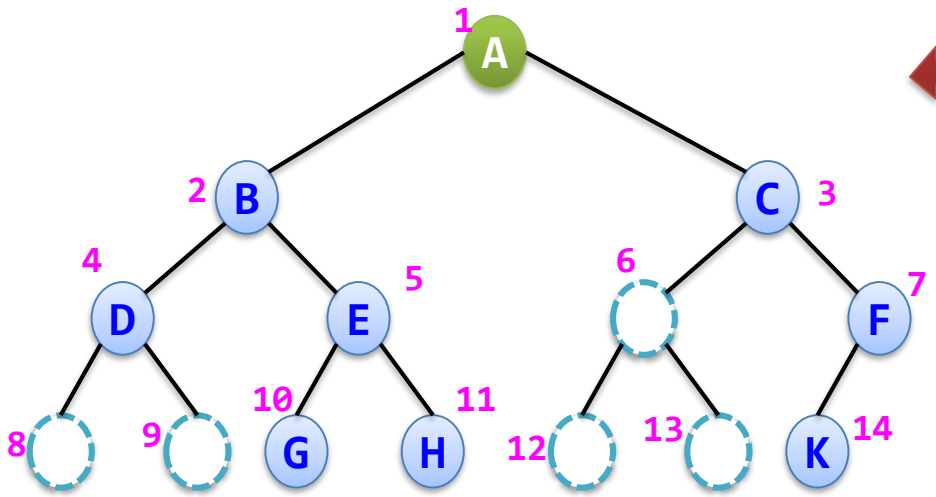
位置	0	1	2	3	4	5	6	7	8	9	10	11	12	13	…
sb	A	B	C	D	E	F	G	H	I	J	K	#	#	#	#

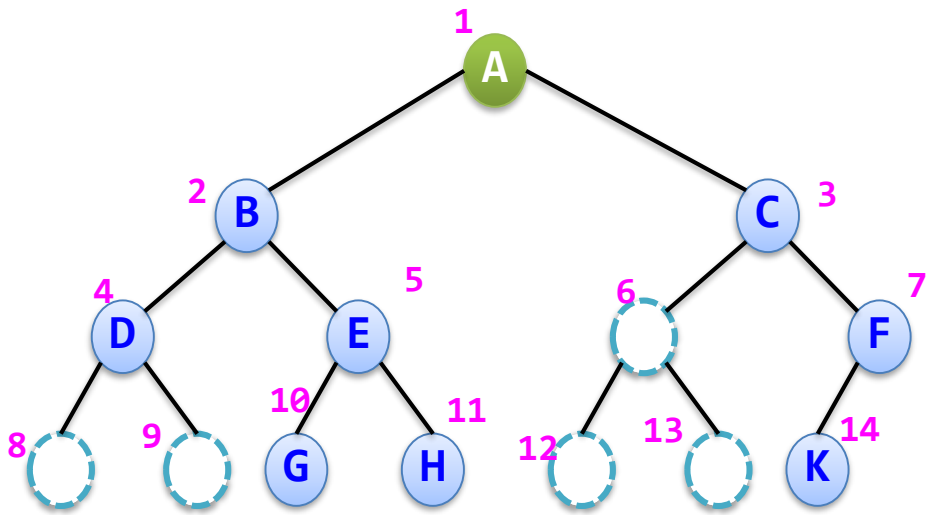
sb的下标从0开始

一般的二叉树 的顺序存储结构



- 增添空结点补齐为一棵完全二叉树
- 并对所有结点进行编号



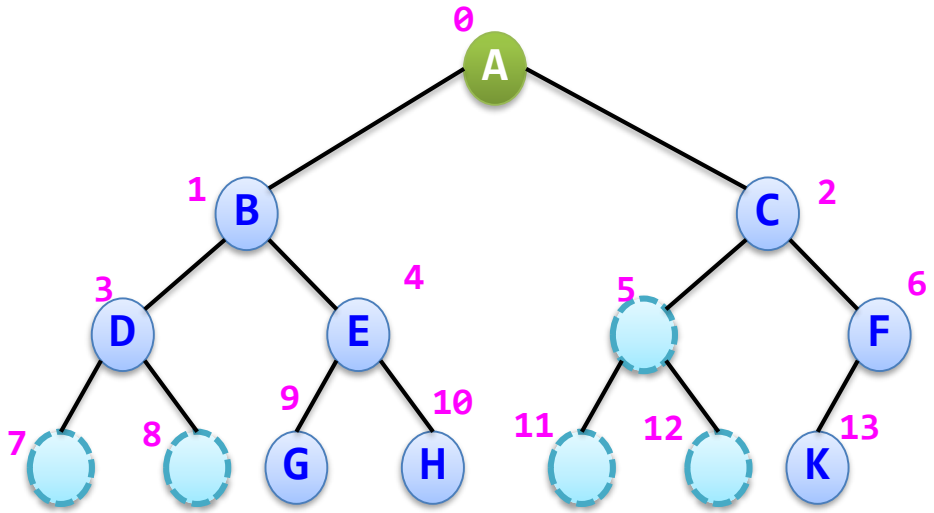


仅保留实际存在的结点值，其他为空

位置	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
sb	A	B	C	D	E	#	F	#	#	G	H	#	#	K	#

sb的下标从1开始

或者



仅保留实际存在的结点值，其他为空

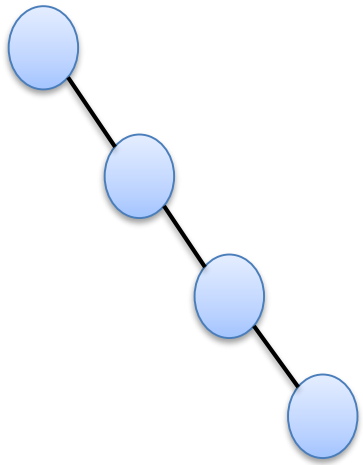
位置	0	1	2	3	4	5	6	7	8	9	10	11	12	13	…
sb	A	B	C	D	E	#	F	#	#	G	H	#	#	K	#

sb的下标从0开始

- 二叉树顺序存储结构采用列表（数组）存放。
- 当二叉树中某结点为空结点或无效结点（不存在该编号的结点）时，对应位置的值用特殊值（如'#'）表示。

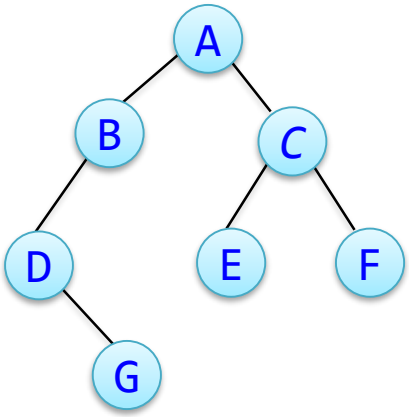
优缺点

- 完全二叉树或满二叉树采用顺序存储结构比较合适
- 如果需要增加很多空结点才能将一棵二叉树改造成为一棵完全二叉树，采用顺序存储结构会造成空间的大量浪费，这时不宜用顺序存储结构。

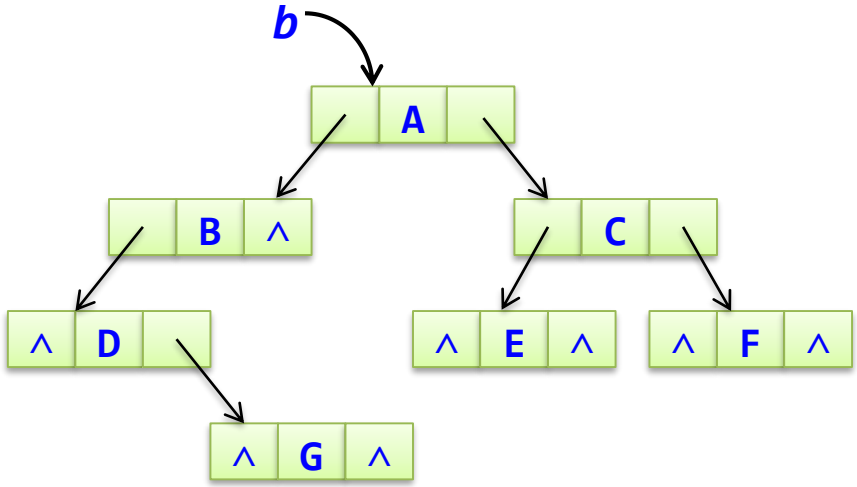


- $h=4$
- $MazSize=2^4-1=15$
- 空间利用率=4/15=27%

2．二叉树的链式存储结构



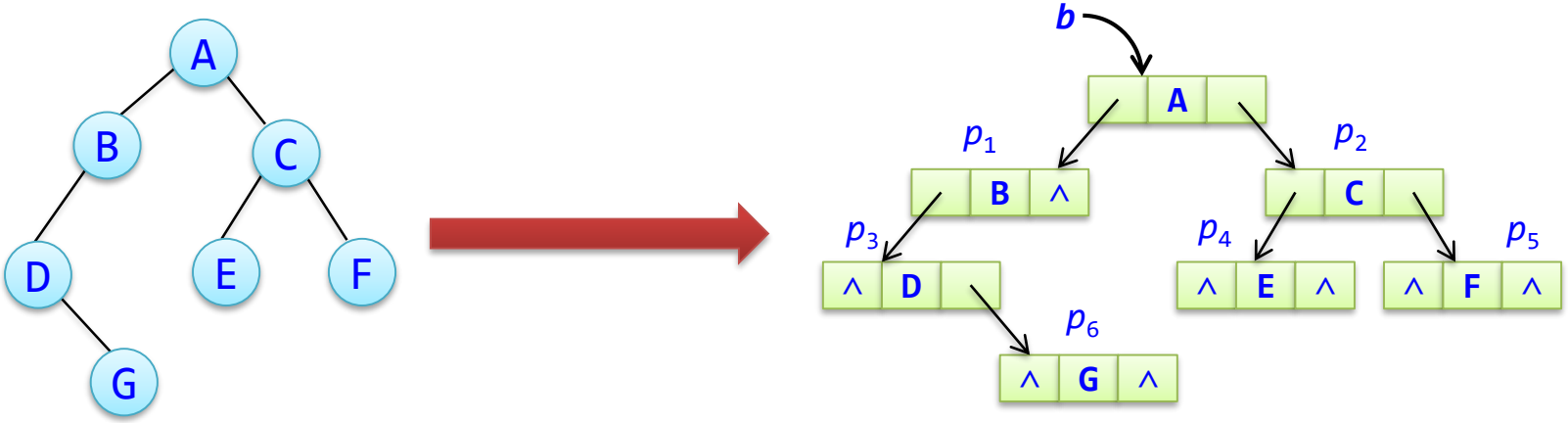
二叉链存储结构



Lchild	Data	Rchild

对应Python语言的二叉链结点类BTNode

```
class BTNode:                          #二叉链中结点类
  def __init__(self,d=None):           #构造方法
    self.data=d                        #结点值
    self.lchild=None                   #左孩子指针
    self.rchild=None                   #右孩子指针
```



建立代码：

```
b=BTNode('A')                #建立各个结点
p1=BTNode('B')
p2=BTNode('C')
p3=BTNode('D')
p4=BTNode('E')
p5=BTNode('F')
p6=BTNode('G')
b.lchild=p1                  #建立结点之间的关系
b.rchild=p2
p1.lchild=p3
p3.rchild=p6
p2.lchild=p4
p2.rchild=p5
```

性质5 含有n个结点的二叉链表中，有n+1个空链域。

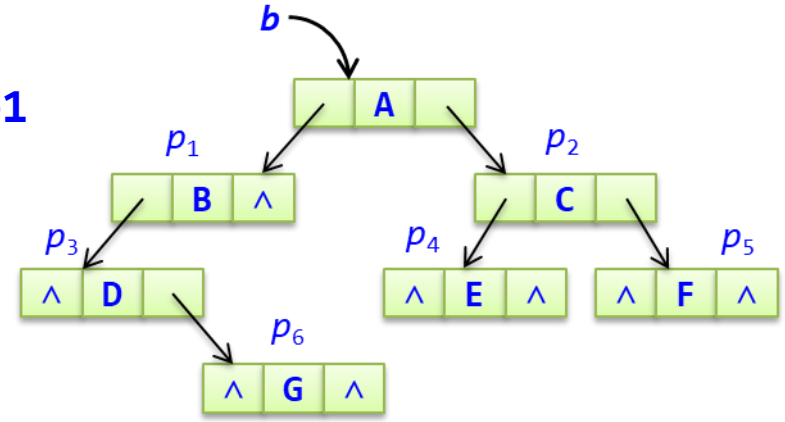
证法一：空链域数=$2n_0+n_1=n_0+n_1+n_0$ （1）

又 $n_0=n_2+1$

故（1）式中的空链域数= $n_0+n_1+n_2+1$

又因为$n=n_0+n_1+n_2$

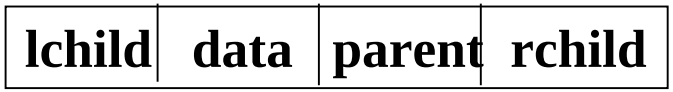
即空链域数=n+1

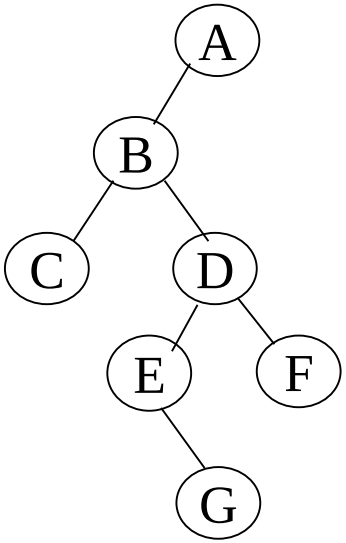


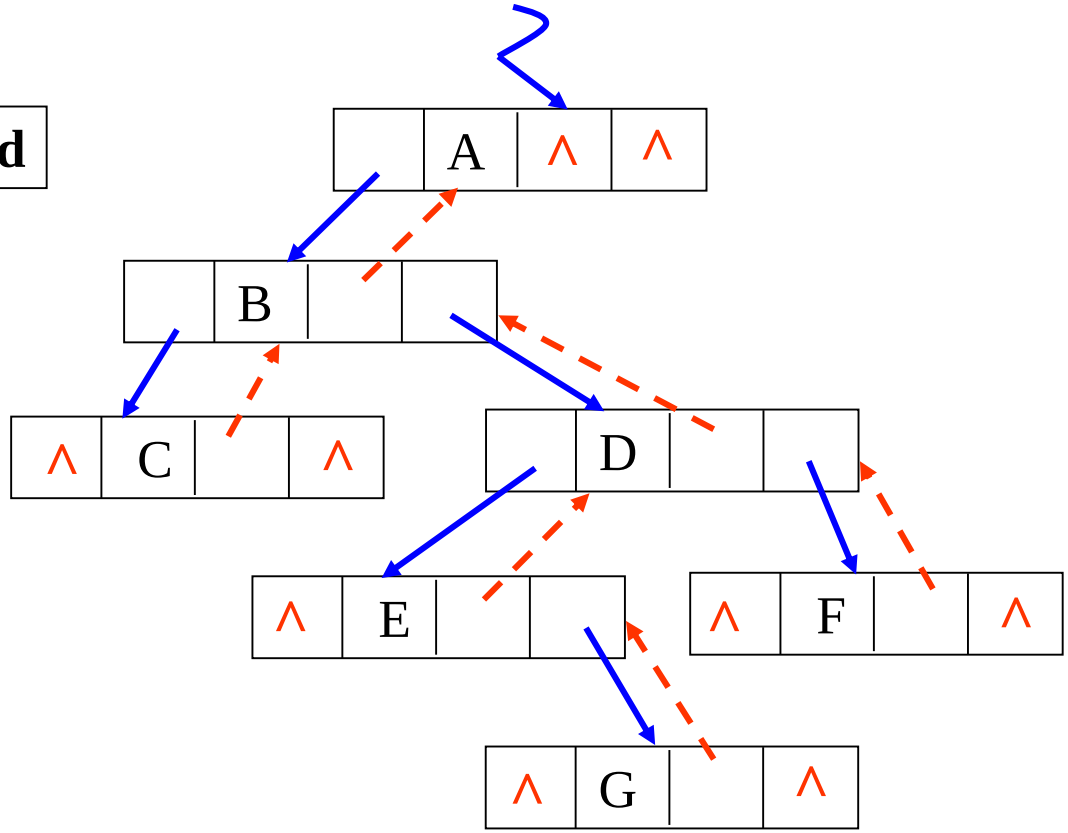
证法二：有n-1个结点有链引入，故非空链域数为：n-1，

而n个结点共有2n个链域，故空链域数为：n+1.

三叉链表



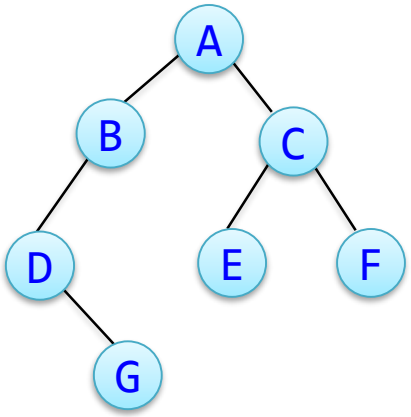




3．二叉树的列表存储结构

每个结点为“[结点值，左子树列表，右子树列表]”的嵌套，当左或者右子树为空时取值None。类型如下：

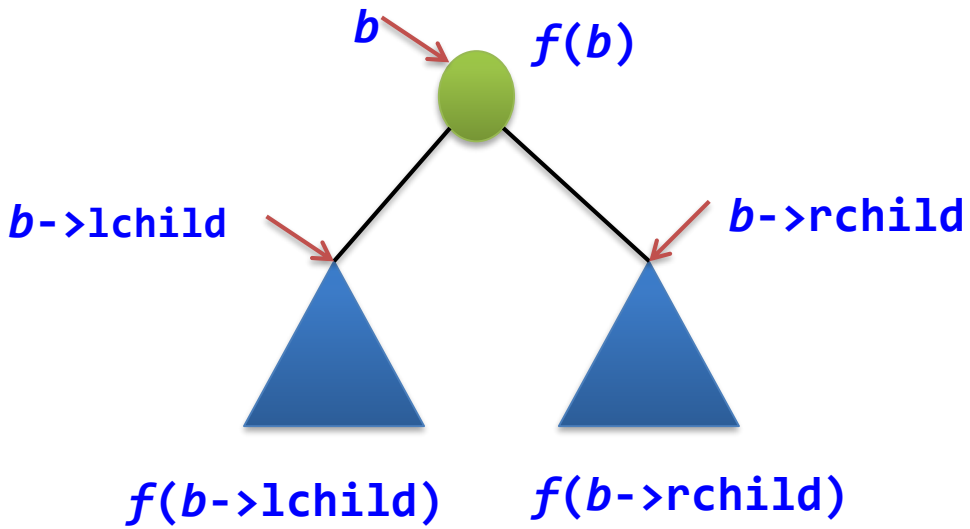
```
class LNode                          #列表存储结构的结点类
  def __init__(self,d=None):         #构造方法
    self.data=d                      #结点值
    self.lchild=None                 #左子树列表
    self.rchild=None                 #右子树列表
```




```
t=['A',
     ['B',['D',None,['G',None,None]]],
     ['C',['E',None,None],['F',None,None]]
  ]
```

6.2.4 二叉树的递归算法设计

一般地，二叉树的递归结构如下：



- 对于二叉树b，设f(b)是求解的“大问题”。
- f(b->lchild)和f(b->rchild)为“小问题”。
- 假设f(b->lchild)和f(b->rchild)是可求的，在此基础上得出f(b)和f(b->lchild)、f(b->rchild)之间的关系，从而得到递归体。
- 再考虑b=NULL或只有一个结点的特殊情况，从而得到递归出口。

例如，假设二叉树中所有结点值为整数，采用二叉链存储结构，求该二叉树b中所有结点值之和。

- 设f(b)为二叉树b中所有结点值之和。
- 则f(b.lchild)和f(b.rchild)分别求根结点b的左、右子树的所有结点值之和。
- 显然有f(b)=b.data+f(b.lchild)+f(b.rchild)。
- 当b=None时f(b)=0，从而得到以下递归模型：

$$
\begin{cases} f(b)=0 & \text{当 } b=\text{None} \\ f(b)=b.data+f(b.lchild)+f(b.rchild) & \text{其他情况} \end{cases}
$$

```
def fun(b):                    #计算以b为根的二叉树的结点值之和
  if b==None: return 0;
  else: return b.data+fun(b.lchild)+fun(b.rchild)
```

6.2.5 二叉树的基本运算及其实现

为了简单，本节讨论的二叉树中所有结点值为单个字符。逻辑结构采用括号表示串，存储结构采用二叉链。

1．二叉树类设计

```
class BTree:                          #二叉树类
  def __init__(self,d=None):          #构造方法
    self.b=None                       #根结点指针
  #二叉树基本运算算法
```

2.二叉树的基本运算算法实现

（1）设置二叉树的根结点SetRoot(b)

```
def SetRoot(self,r):                    #设置根结点为r
  self.b=r
```

（2）求二叉链的括号表示串DispBTree()

```
def DispBTree(self):                          #返回二叉链的括号表示串
  return self._DispBTree(self.b)

def _DispBTree(self,t):                       #被DispBTree方法调用
  if t==None:                                 #空树返回空串
    return ""
  else:
    bstr=t.data                               #输出根结点值
    if t.lchild!=None or t.rchild!=None:
      bstr+="("                               #有孩子结点时输出"("
      bstr+=self._DispBTree(t.lchild)         #递归输出左子树
      if t.rchild!=None:
        bstr+=","                             #有右孩子结点时输出","
      bstr+=self._DispBTree(t.rchild)         #递归输出右子树
      bstr+=")"                               #输出")"
  return bstr
```

（3）查找值为x的结点FindNode(x)

设*f*(*t*，*x*)在以*t*为根结点的二叉树中查找值为*x*的结点，找到后返回其地址，否则返回None。

f(*t*，*x*)=None	若*t*=None
f(*t*，*x*)=t	若t.data=*x*
f(*t*，*x*)=*p*	若在左子树中找到了，即 *p*=*f*(t.lchild，*x*)且p!=None
f(*t*，*x*)=*f*(t.rchild，*x*)	其他情况

```
def FindNode(self,x):                          #查找值为x的结点算法
  return self._FindNode(self.b,x)

def _FindNode(self,t,x):                       #被FindNode方法调用
  if t==None:
     return None                               #t为空时返回None
  elif t.data==x:
     return t                                  #t所指结点值为x时返回t
  else:
     p=self._FindNode(t.lchild,x)              #在左子树中查找
     if p!=None:
        return p                               #在左子树中找到p结点，返回p
     else:
        return self._FindNode(t.rchild,x)      #返回在右子树中查找结果
```

（4）求高度Height()

设以*t*为根结点二叉树的高度为*f*(*t*)，空树高度为0，非空树高度为左、右子树中较大的高度加1。

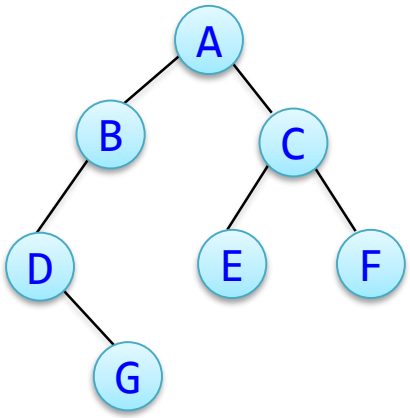
$$f(t) = 0 \qquad \text{若t=None}$$

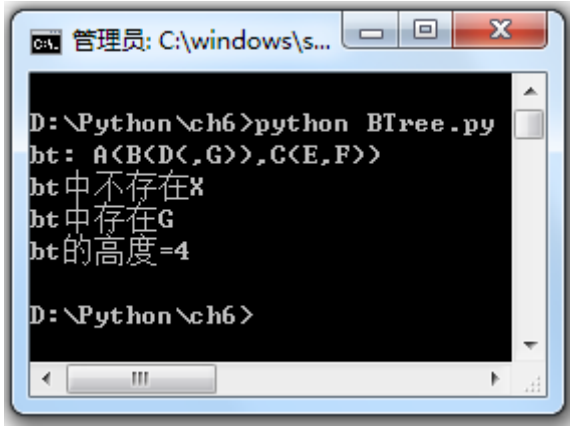
$$f(t) = MAX\{f(t.lchild), f(t.rchild)\}+1 \qquad \text{其他情况}$$

```
def Height(self):                          #求二叉树高度的算法
    return self._Height(self.b)

def _Height(self,t):                       #被Height方法调用
  if t==None:
     return 0                              #空树的高度为0
  else:
     lh=self._Height(t.lchild)             #求左子树高度lchildh
     rh=self._Height(t.rchild)             #求右子树高度rchildh
     return max(lh,rh)+1
```

程序验证

```
if __name__ == '__main__':
  b=BTNode('A')              #建立各个结点
  p1=BTNode('B');p2=BTNode('C');p3=BTNode('D')
  p4=BTNode('E');p5=BTNode('F');p6=BTNode('G')
  b.lchild=p1                #建立结点之间的关系
  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='X';p=bt.FindNode(x)
  if p!=None: print("bt中存在"+x)
  else: print("bt中不存在"+x)

  x='G';p=bt.FindNode(x)
  if p!=None: print("bt中存在"+x)
  else: print("bt中不存在"+x)

  print("bt的高度=%d" %(bt.Height()))
```





操作

所有代码存放在BTree.py中