数据结构与算法B

06-二叉树

内容提要

- 树和二叉树基本概念
- 二叉树
  - 存储结构
  - 周游算法
  - 建立一棵二叉树
- 二叉树的应用
  - 哈夫曼树
  - 二叉检索树/排序树
- 树与树林

树的例子

- 本章我们来讨论一种基本的“非线性”数据结构——树；
- 树在计算机科学的各个领域中被广泛应用
  - 操作系统、图形学、数据库管理系统、计算机网络
- 跟自然界中的树一样，数据结构树也分为：根、枝和叶等三个部分
  - 但一般数据结构的图示把根放在上方
  - 叶放在下方

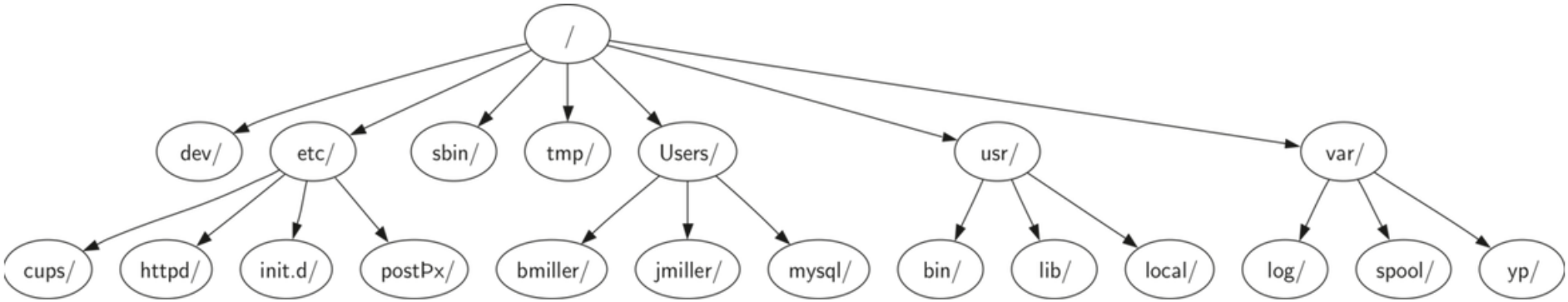
树的例子：生物学物种分类体系

- 首先我们看到分类体系是层次化的
  - 树是一种分层结构
  - 越接近顶部的层越普遍
  - 越接近底部的层越独特
  - 界、门、纲、目、科、属、种
- 分类树的用法：辨认物种
  - 从顶端开始，沿着箭头方向向下
  - 门：脊索动物还是节肢动物？
  - 纲：哺乳动物么？
  - 目：食肉动物？
  - 科：猫科？
  - 属：猫属？
  - 种：家猫！

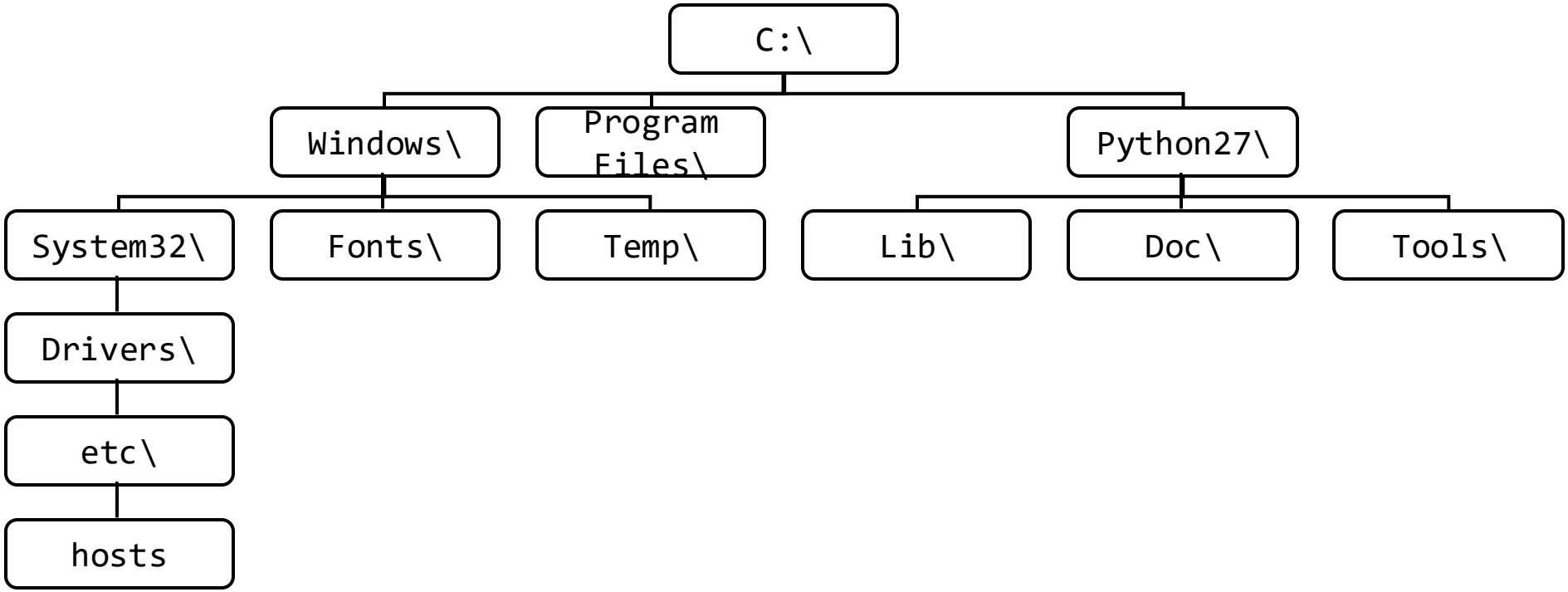
树的例子：生物学物种分类体系

- 分类树的第二个特征：一个节点的子节点与另一个节点的子节点相互之间是隔离、独立的
  - 猫属Felis和蝇属Musca下面都有Domestica的同名节点
  - 但相互之间并无任何关联，可以修改其中一个Domestica而不影响另一个。
- 分类树的第三个特征：每一个叶节点都具有唯一性
  - 可以用从根开始到达每个种的完全路径来唯一标识每个物种
  - 动物界->脊索门->哺乳纲->食肉目->猫科->猫属->家猫种
  - Animalia->Chordate->Mammal->Carnivora->Felidae->Felis->Domestica

树的例子：文件系统：Linux

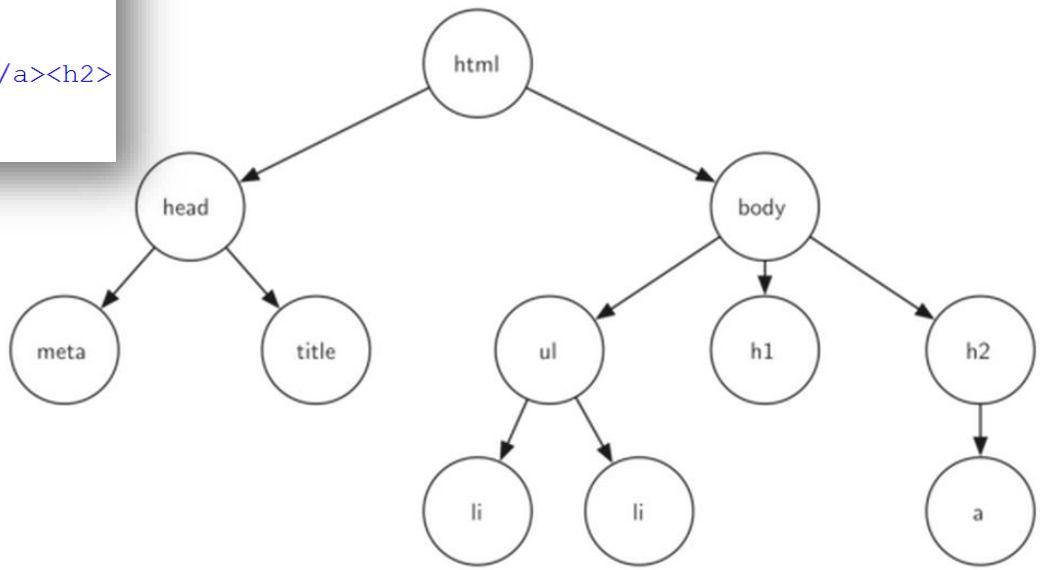


树的例子：文件系统：Windows

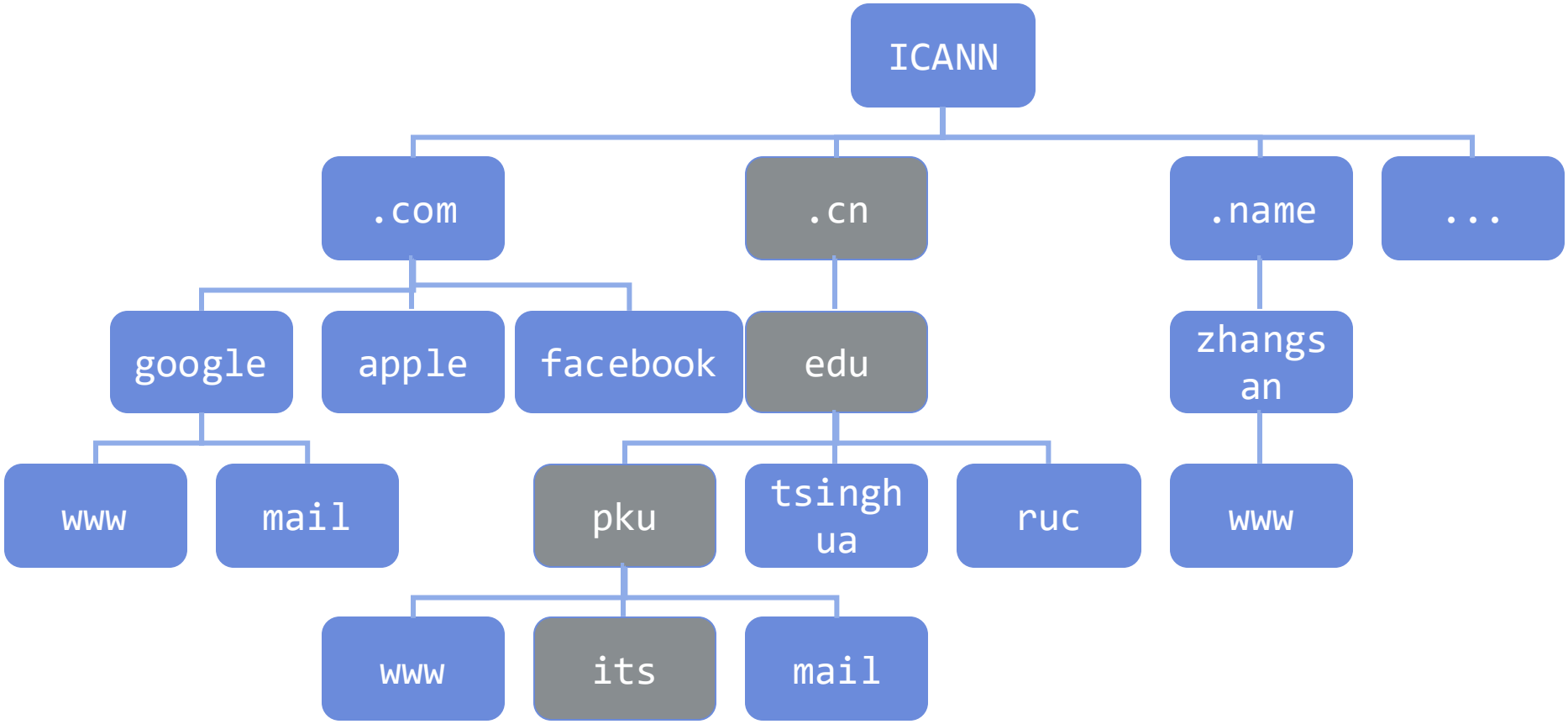


树的例子：HTML文档（嵌套标记）

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type"
          content="text/html; charset=utf-8" />
    <title>simple</title>
</head>
<body>
<h1>A simple web page</h1>
<ul>
    <li>List item one</li>
    <li>List item two</li>
</ul>
<h2><a href="http://www.cs.luther.edu">Luther CS </a><h2>
</body>
</html>
```

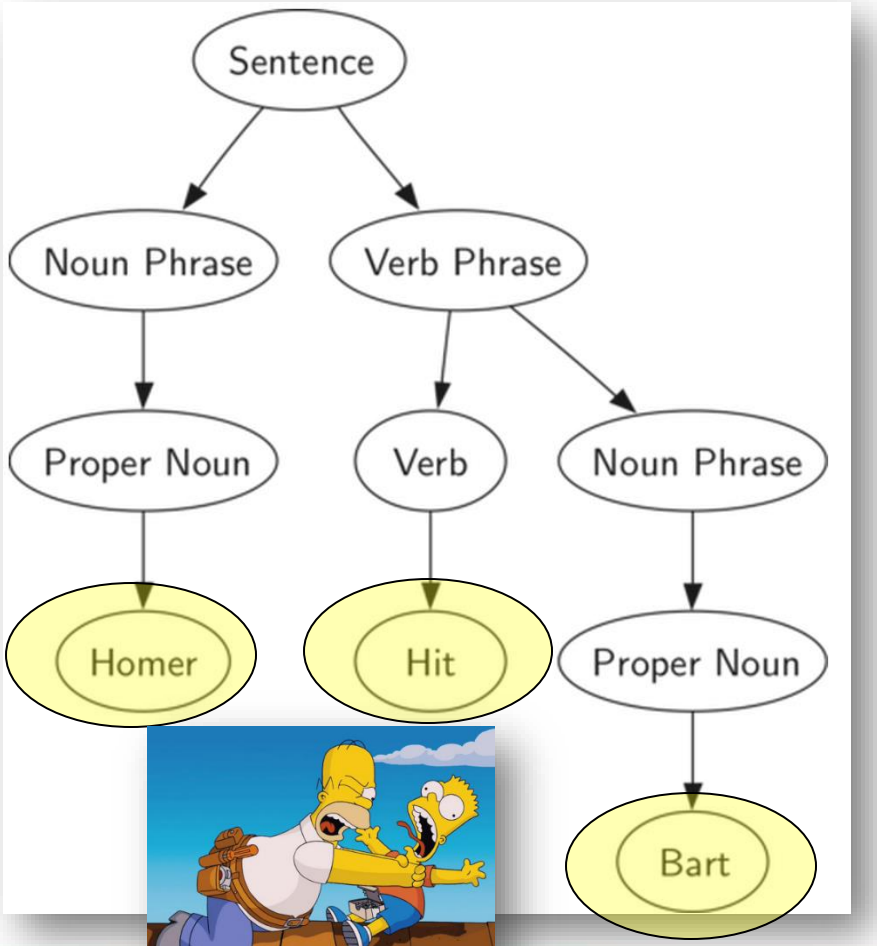


树的例子：域名体系


ICANN
.com
.cn
.name
...
google
apple
facebook
edu
zhangsan
www
mail
pku
tsinghua
ruc
www
www
its
mail

树的应用：解析树Parse Tree（语法树）

- 将树用于表示语言中句子，可以分析句子的各种语法成分，对句子的各种成分进行处理
- 语法分析树
  - 主谓宾，定状补
- 程序设计语言的编译
  - 词法、语法检查
  - 从语法树生成目标代码
- 自然语言处理
  - 机器翻译、语义理解



树的应用：解析树Parse Tree（表达式树）

- 我们还可以将表达式表示为树结构
  - 叶节点保存操作数，内部节点保存操作符
- 全括号表达式((7+3)*(5-2))
  - 由于括号的存在，需要计算*的话，就必须先计算7+3和5-2
  - 表达式树的层次帮助我们了解表达式计算的优先级
  - 越底层的表达式，优先级越高
- 树中每个子树都表示一个子表达式
  - 将子树替换为子表达式值的节点，即可实现求值

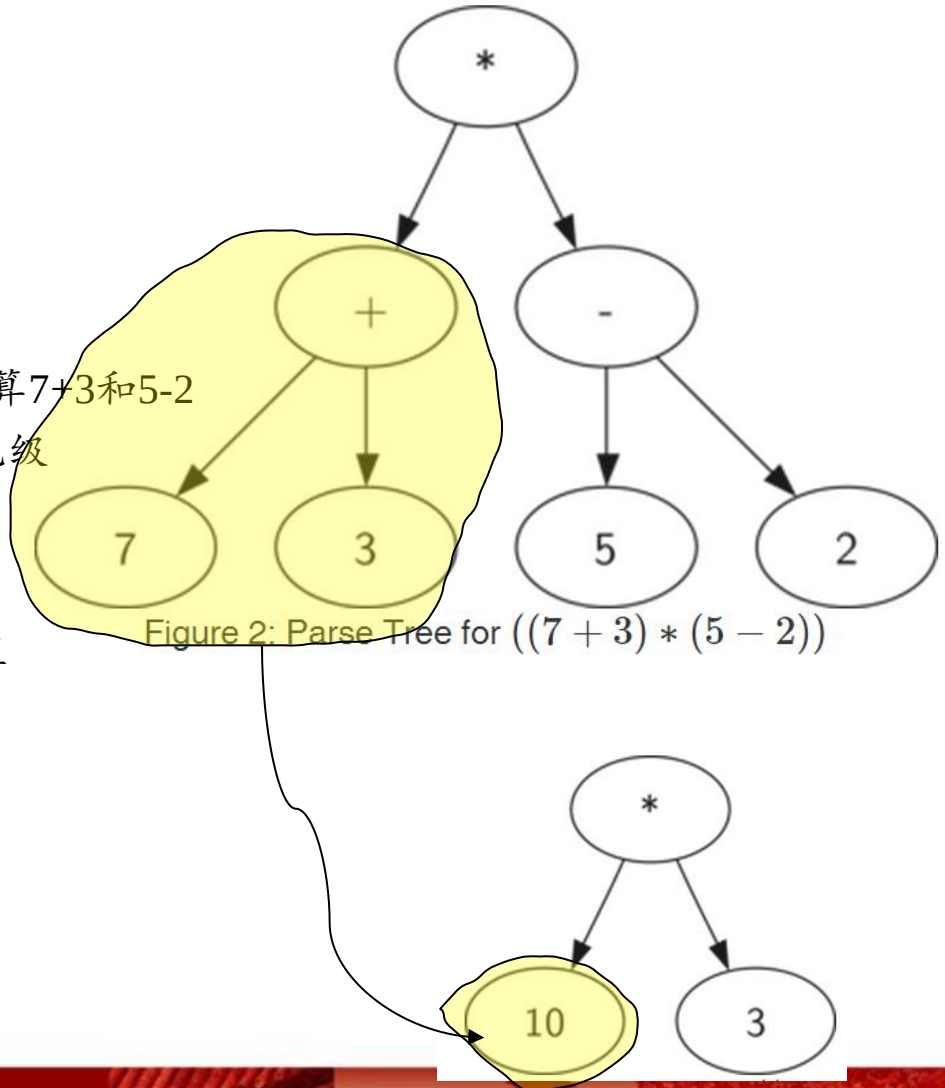


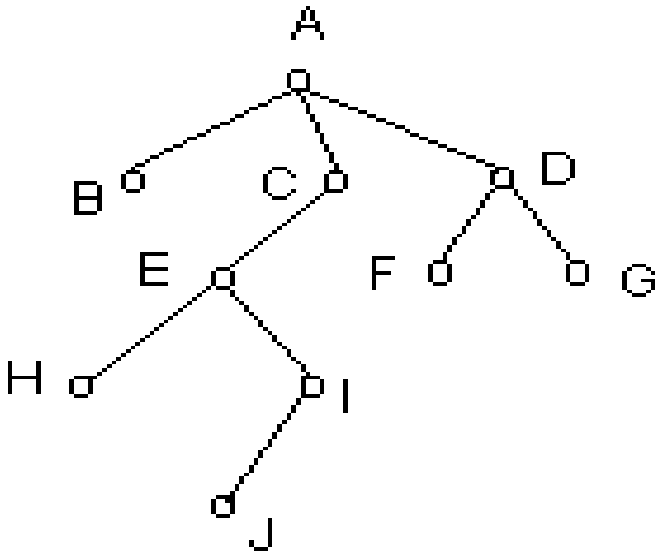
Figure 2: Parse Tree for $((7+3)*(5-2))$

树的逻辑结构

- 包含n个结点的有穷集合K (n>0)，且在K上定义了一个关系 r，关系 r 满足以下条件：
  - 有且仅有一个结点$k_0 \in K$，它对于关系 r 来说没有前驱。结点k_0称作树的根；
  - 除结点k_0外，K中的每个结点对于关系 r 来说都有且仅有一个前驱；
  - 除结点k_0外的任何结点$k \in K$，都存在一个结点序列k_0，k_1，…，k_s，使得k_0就是树根，且$k_s=k$，其中有序对$<k_{i-1}, k_i> \in r$ $(1 \le i \le s)$。这样的结点序列称为从根到结点k的一条路径。

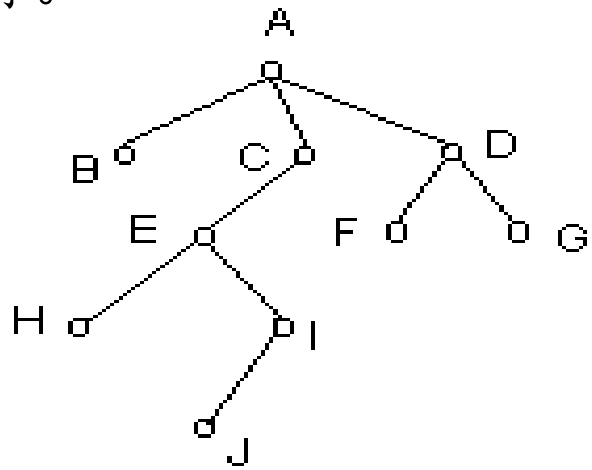
树的逻辑表示

- **T = (N, R)**
  - **结点集合 N = { A, B, C, D, E, F, G, H, I, J}**
  - **N上的关系 R = { <A, B>, <A, C>, <A, D>, <C, E>, <D, F>, <D, G>, <E, H>, <E, I>, <I, J> }**



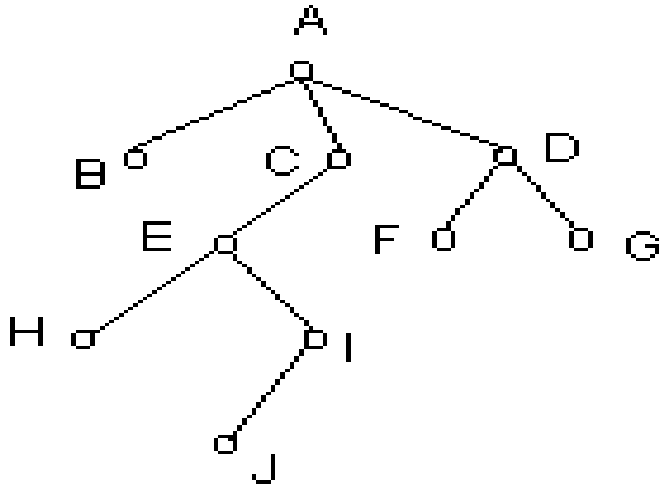
基本术语-1

- 父结点、子结点、边
  - –若结点y是结点x的一棵子树的根，则x称作y的“父结点”（或父母）；
  - –y称作x的“子结点”（或子女）；
  - –有序对<x，y>称作从x到y的“边”
- 兄弟
  - –具有同一父母的结点彼此称作“兄弟”
  - –树t中B，C，D互为兄弟，F，G互为兄弟，等等。
  - –注意，E和F并不是兄弟



基本术语-2

- 路径、路径长度
  - 对二叉树上任意两个结点V_i，V_j，必然存在唯一的不重复的结点序列$\{V_i, P_1, P_2, \ldots, P_m, V_j\}$，使得$V_i$和$P_1$之间有边，$P_m$和$V_j$之间有边，$P_k$和$P_{k+1}$之间有边，这个结点序列就被称作为$V_i$到$V_j$的**路径**。
  - 路径上边的数量被称作是**路径长度**。
- 祖先、子孙
  - 若结点y在以结点x为根的一个子树（或树）中，且y≠x，称x是y的“祖先”，y是x的“子孙”



基本术语-3

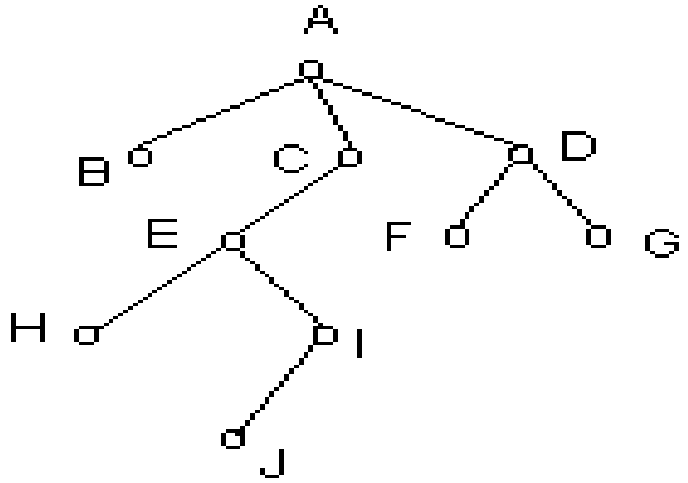
⑩结点的层数、树的深度

–**层数**：根为第 0 层

•其他结点的层数等于其父结点的层数加 1

–**深度/高度：层数最大的叶结点的层数**

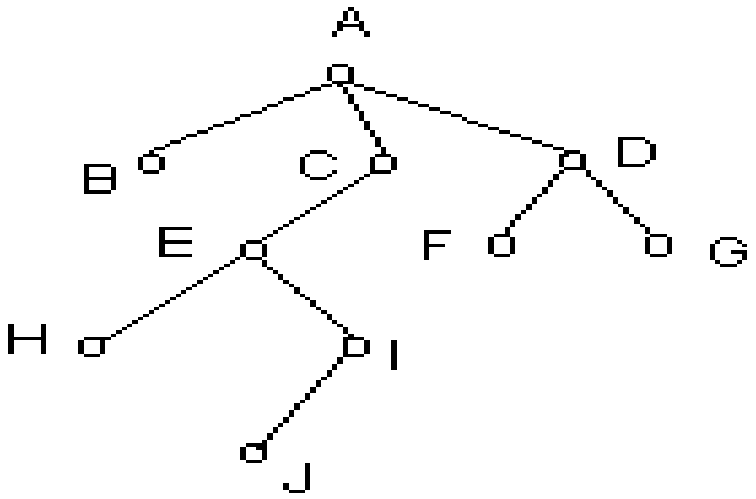
•也有些定义是上面的数值+1（取决于是否把根节点算上）



基本术语-4

结点的度数、树的度数

–结点的子女个数叫作结点的“度数”。树中度数最大的结点的度数叫作“树的度数”



树叶、分支结点

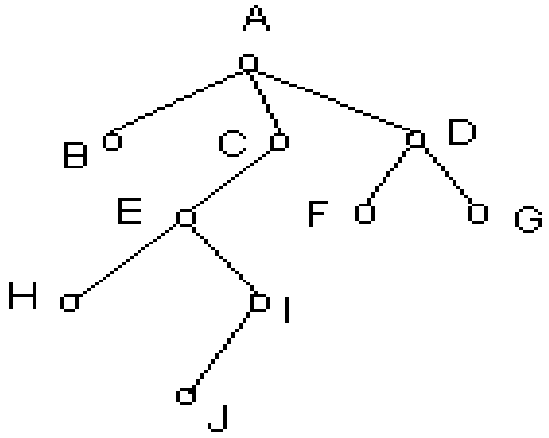
–度数为0的结点称作“树叶”（又叫终端结点）

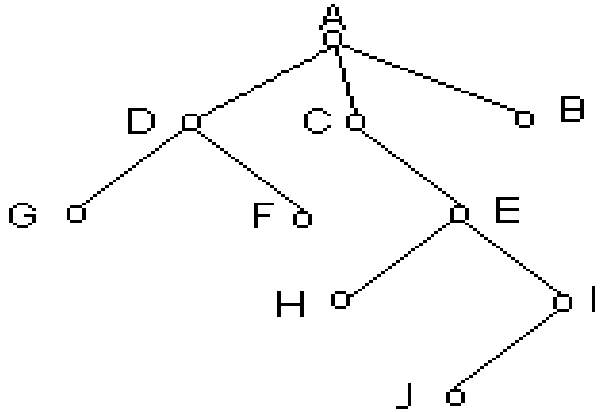
–度数大于0的结点称作“分支结点”或非终端结点

❖注意，结点的度数为1时，虽然只有一个子女，也叫分支结点。这两个术语对于根结点也不例外

基本术语-5

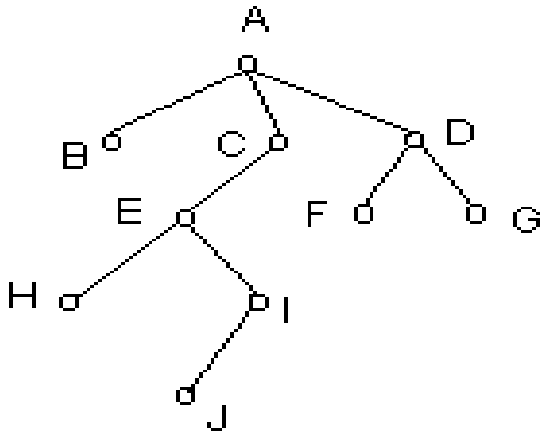
- 无序树、有序树
  - 对子树的次序不加区别的树叫作“无序树”。对子树之间的次序加以区别的树叫作“有序树”
  - 例如在下图中，按无序树的概念t和t’是同一棵树，按有序树的概念则是不同的树，**本章讨论的树一般是有序树**





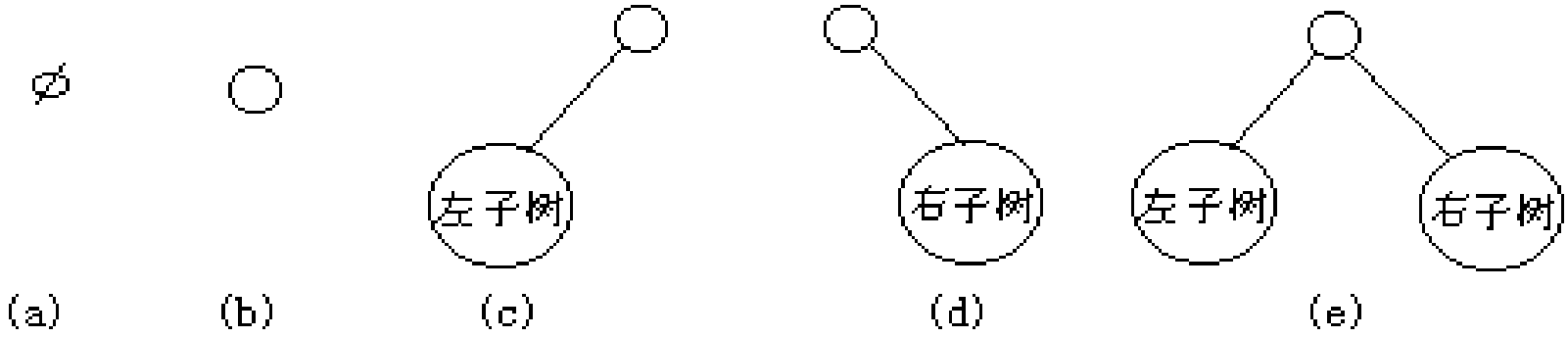
基本术语-6

- 结点的次序
  - –在有序树中可以从左到右地规定结点的次序
  - –例如下图中，结点B，C，D是从左到右排序的；可以说结点C是结点B右边的结点，是结点D左边的结点
  - –结点C的所有子女都在结点B及其子女的右边，而在结点D及其子女的左边
  - –按从左到右的顺序，可以把一个结点的最左边的结点简称“长子”，长子右边的结点称为“次子”，…



二叉树

- 递归定义：
  - 结点的有限集合，这个集合或者为**空集**，或者由一个**根**及两棵不相交的分别称作这个根的“**左子树**”和“**右子树**”的二叉树组成。
- 二叉树的五种基本形态

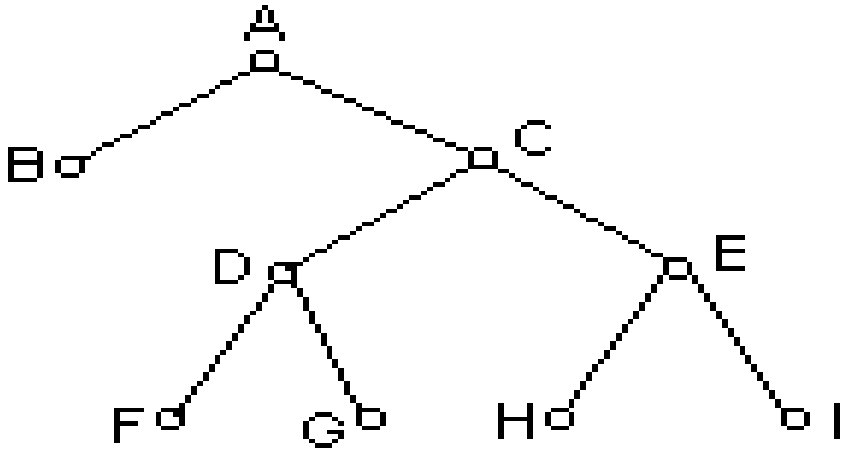


二叉树 vs 树

- 二叉树的术语与树中的相应部分类似
- 二叉树不是树的特殊情形，它们是两种数据结构
- 树和二叉树之间最主要的差别：
  - 二叉树中结点的子树要区分为左子树和右子树，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
  - 譬如上例中，（c）和（d）是两棵不同的二叉树，但作为树，它们是相同的

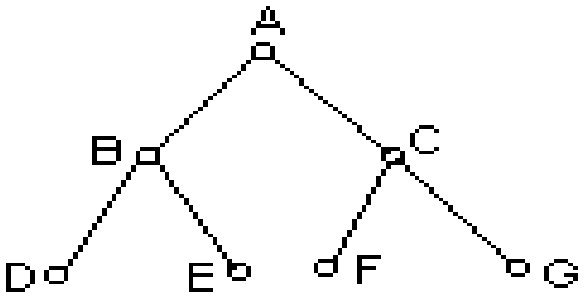
满二叉树（Full Binary Tree）

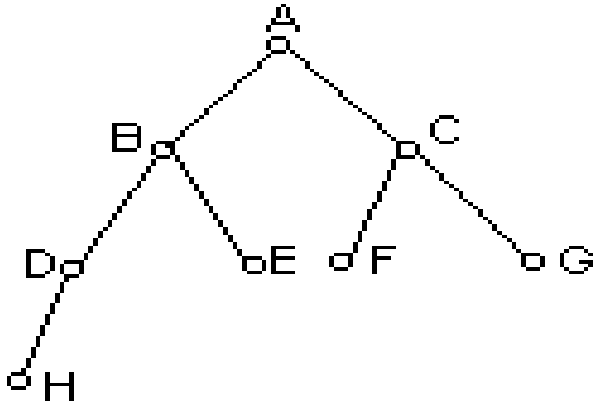
- 如果一棵二叉树的任何结点，或者是树叶，或者恰有两棵非空子树，则此二叉树称作满二叉树



完全二叉树（Complete Binary Tree）

- 若一棵二叉树，最多只有最下面的两层结点度数可以小于2，且最下面一层的结点都集中在该层最左边的若干位置上，则称此二叉树为完全二叉树

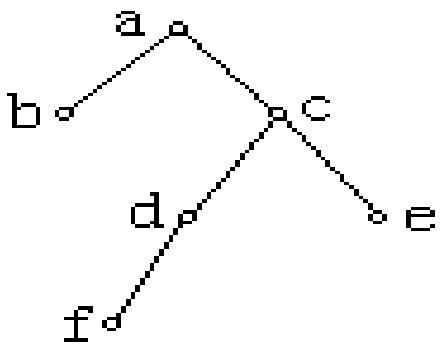


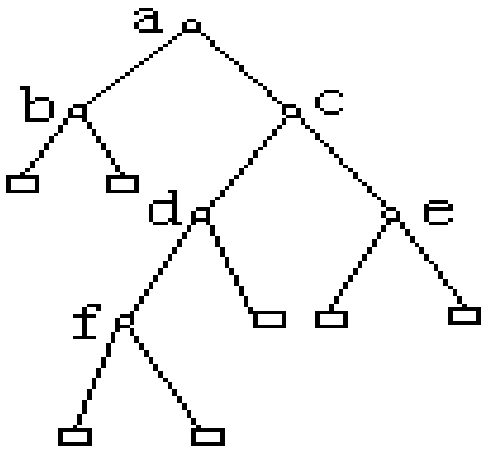


- 在许多算法和算法分析中都明显地或隐含地用到完全二叉树的概念

扩充二叉树（Extended Binary Tree）

- 扩充二叉树：把原二叉树的结点都变为度数为2的分支结点
  - –如果原结点的度数为2，则不变
  - –度数为1，则增加一个分支
  - –度数为0（树叶）增加两个分支





=>满二叉树

扩充二叉树

- 在扩充的二叉树里，新增加的结点（树叶），称为外部结点；原有结点称为内部结点
- “外部路径长度”E：在扩充的二叉树里从根到每个外部结点的路径长度之和
- “内部路径长度”I： 在扩充的二叉树里从根到每个内部结点的路径长度之和

二叉树的基本性质1

性质1 在非空二叉树的第i层上至多有2^i个结点(i≥0)

证明：用归纳法来证。

① 归纳基础：i=0时，显然$2^i = 2^0 = 1$是对的

② 归纳假设：假定对所有的j(0≤j≤i)，命题成立，即第j层上至多有2^j个结点

③ 归纳推理：下面证明当j=i+1时，命题也成立

由归纳假设可知：第i层上至多有2^i个结点。又由于二叉树中每个结点的度至多为2，所以第i+1层上的最大结点个数是第i层上最大结点个数的2倍，即$2*2^i = 2^{i+1}$

结论：故二叉树的第i层上至多有2^i个结点(i≥0)

二叉树的基本性质2

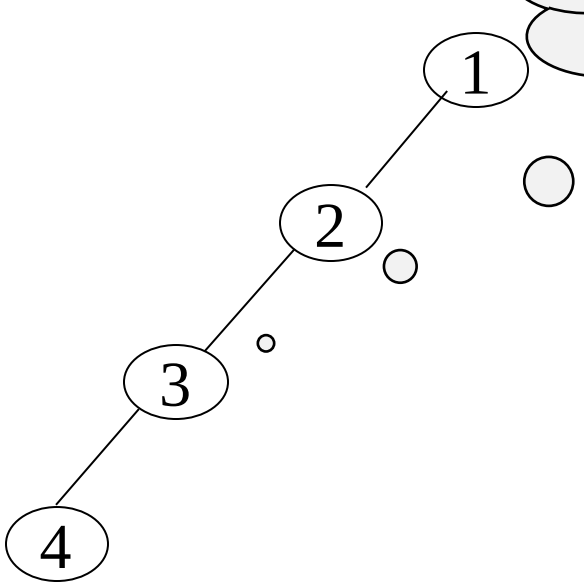
性质2 深度为k的二叉树中最多有$2^{k+1}-1$个结点($k\geq 0$)

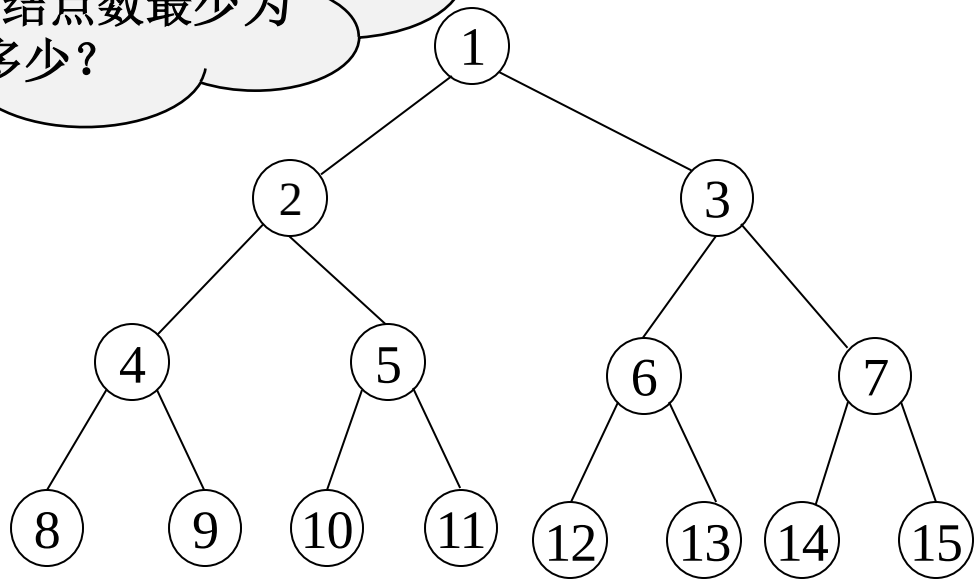
证明：假设第i层上的最大结点个数是m_i，由性质1可知，深度为k的二叉树中最大结点个数M为:

$$M=\sum_{i=0}^{k} m_i \leq \sum_{i=0}^{k} 2^i = 2^{k+1}-1$$

示例

深度为h的二叉树上只有度为0和2的结点，那么二叉树结点数最少为多少？





结点最少的二叉树：

$1+1+1+1$

结点最多的二叉树：

$2^0+2^1+2^2+2^3$

二叉树的基本性质3

性质3 对于任何一棵非空的二叉树，如果叶结点个数为n_0，度为2的结点个数为n_2，则有$n_0 = n_2 + 1$

证明：

设一棵非空二叉树中有n个结点，度为1的结点个数为n_1，则$n = n_0 + n_1 + n_2$ (1)

在二叉树中，除根结点外，其余每个结点都有一个分支进入，假设B为分支总数，则有$B = n - 1$ (2)

二叉树中的分支都是由度为1和2的结点发出的，有 $B = n_1 + 2n_2$ (3)

综合(1)、(2)、(3)式可得

$$n_0 = n_2 + 1$$

二叉树的基本性质4

性质4.　具有n个结点的完全二叉树的深度k为 $\lfloor \log_2 n \rfloor$

证明：$n = 2^0 + 2^1 + 2^2 + \ldots + 2^{k-1} + m_k$

$= 2^k - 1 + m_k$

$2^k - 1 < n \leq 2^{k+1} - 1$　（性质2）

$2^k \leq n < 2^{k+1}$

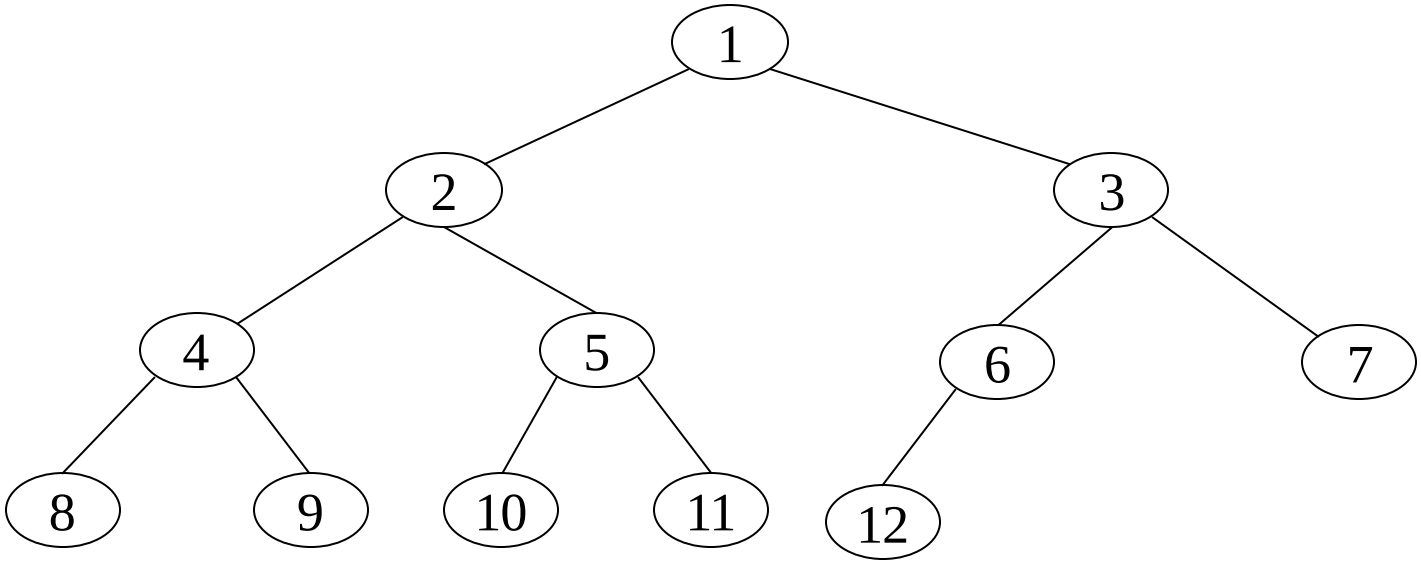
$k \leq \log_2 n < k+1$

$\therefore \quad k = \lfloor \log_2 n \rfloor$

示例

- 完全二叉树层次序列反映出它的结构

1 2 3 4 5 6 7 8 9 10 11 12



二叉树的基本性质5

性质5 对于具有n个结点的完全二叉树，如果按照从上到下和从左到右的顺序对树中的所有结点从1开始进行编号，则对于任意的序号为 i 的结点，有：

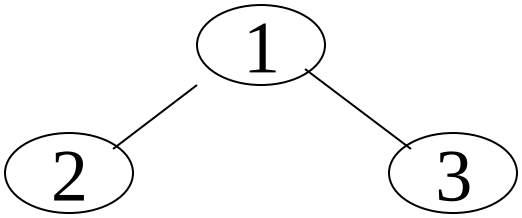
- 如果i＞1，则其父结点的序号为 int (i/2)；
  如果i = 1，则其是根结点，它没有父结点。
- 如果2i≤n，则其左子女结点的序号为2i；
  如果2i＞n，则其没有左子女结点。
- 如果2i+ 1 ≤n，则其右子女结点的序号为2i+1；
  如果2i+ 1 ＞n，则其没有右子女结点。

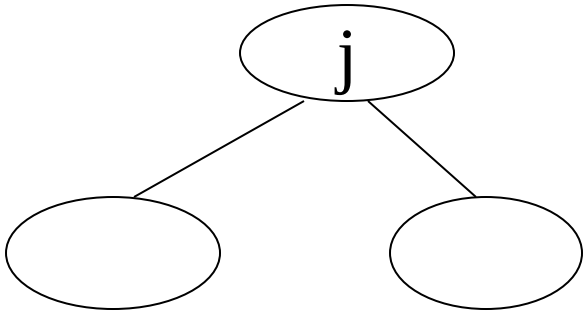
二叉树的基本性质5---证明

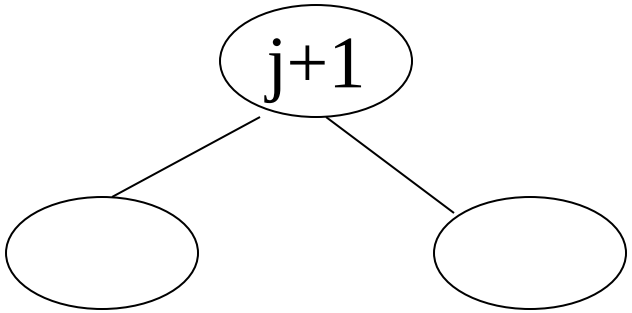
1. 归纳基础：对于（2）和（3）当i=1时，若$2i = 2 \le n$，左子女结点的序号为2。$2i+1 = 3 \le n$，右子女结点的序号为3。
2. 归纳假设：假设对于序号为j的结点，命题成立。
3. 归纳推理：对于i=j+1，
   - 其左子女结点的序号等于j的右子女结点的序号加1，即 2j+1+1=2(j+1)
   - 其右子女结点的序号等于：2(j+1)+1。
4. 根据（2）和（3），可知i的父结点的序号为$\lfloor i/2 \rfloor$.

结论：完全二叉树的层次序列，反映了它的结构。

性质5的示例







二叉树的基本性质6

性质6 在满二叉树中，叶节点的个数比分支节点的个数多1。

证明：

非空满二叉树树叶数等于其分支结点数加1。

二叉树的基本性质7

性质7　在扩充的二叉树里，新增加的外部结点的个数比原来的内部结点个数多1。

证明

–对于任何一棵非空的二叉树，如果叶结点个数为n_0，度为2的结点个数为n_2，度为1的结点个数为n_1，原来的内部结点个数为$n = n_0 + n_1 + n_2$,因为$n_0 = n_2 + 1$，则$n = n_1 + 2n_0 - 1$

–新增加的外部结点为：$m = n_1 + 2n_0$

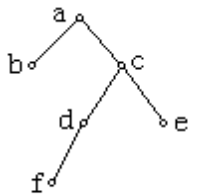
二叉树的基本性质8

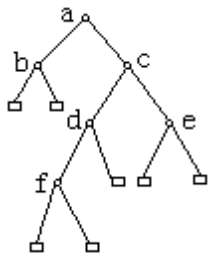
性质8　对任意扩充二叉树，E和I之间满足以下关系：E = I + 2n，其中n是内部结点个数。

譬如

E = 2 + 2 + 4 + 4 + 3 + 3 + 3 = 21

I = 0 + 1 + 1 + 2 + 2 + 3 = 9





证明

1. 归纳基础：当n=1时，I=0, E=2，此等式成立。
2. 归纳假设：设有n个内部结点的扩充二叉树，下式成立。

$$E_n = I_n + 2n \qquad (1)$$

3. 归纳推理：对于 n+1 个内部结点的扩充二叉树，去掉一个原来为树叶、路径长度为K的内部结点，内部路径长度变为：

$$I_n = I_{n+1} - K \qquad (2)$$

外部路径长度变为：

$$E_n = E_{n+1} - 2(K+1) + K = E_{n+1} - K - 2$$

即：

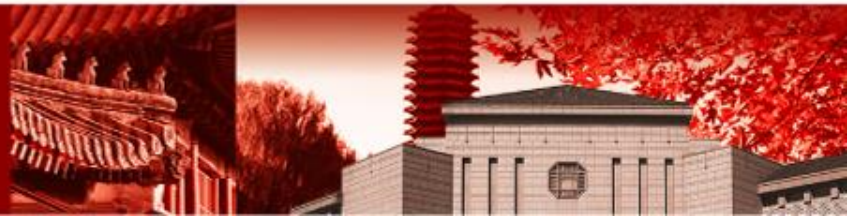
$$E_{n+1} = E_n + K + 2$$

$$E_{n+1} = \underline{(I_n + 2n)} + K + 2 = \underline{(I_{n+1} - K)} + 2n + K + 2$$

代入（1）　　代入（2）

$$= I_{n+1} + 2(n+1)$$

如何存储一个二叉树？



二叉树的实现

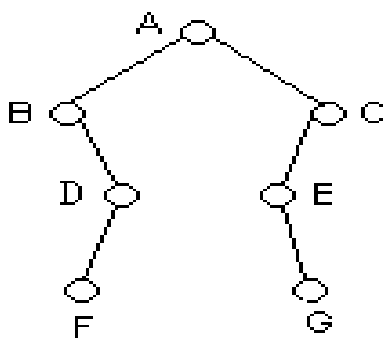
- 根据应用的不同以及二叉树本身的不同特点，二叉树可采用不同的存储结构
  - –顺序表示
  - –链式表示

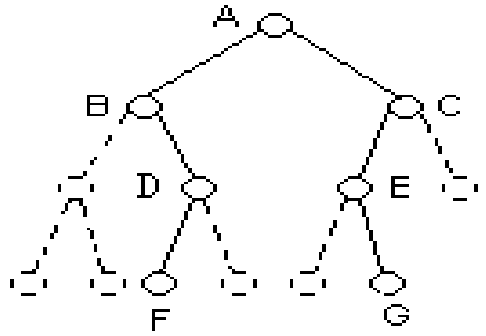
完全二叉树的顺序表示

- 完全二叉树结点的顺序存储
  - 所有结点按层次顺序依次存储在连续的存储单元中，
  - 根据一个结点的存储地址就可算出它的左右子女，父母的存储地址，如同存储了相应的指针一样。
- 顺序表示是存储完全二叉树的最简、最节省空间的存储方式
  - 完全二叉树的顺序存储，在存储结构上是线性的，但在逻辑结构上它仍然是二叉树型结构

一般二叉树的顺序表示

- 增加空结点来构造一棵完全二叉树，再以二叉树的方式存储
- 接近于完全二叉树的形态，需要增加的空结点数目不多，则也可采用顺序表示；





A	B	C	∧	D	E	∧	∧	∧	F	∧	∧	G
数组下标	0	1	2	3	4	5	6	7	8	9	10	11	12

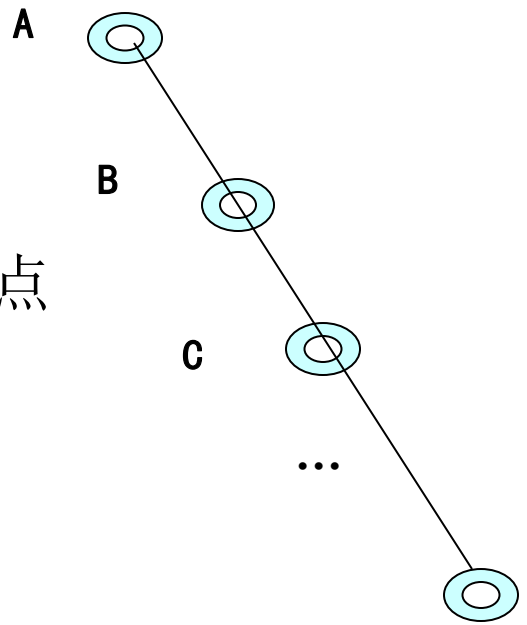
顺序表示的二叉树定义

采用顺序存储表示的二叉树定义如下:

```
class BinaryTree:
    def __init__(self, max_nodes=100):
        self.nodes = [None] * max_nodes
        self.size = 0

    def insert(self, value):
        if self.size < len(self.nodes):
            self.nodes[self.size] = value
            self.size += 1

    def parent(self, index):
        return self.nodes[(index - 1) // 2] if index > 0 else None

    def left_child(self, index):
        left_index = 2 * index + 1
        return self.nodes[left_index] if left_index < self.size else
None

    def right_child(self, index):
        right_index = 2 * index + 2
        return self.nodes[right_index] if right_index < self.size
else None
```

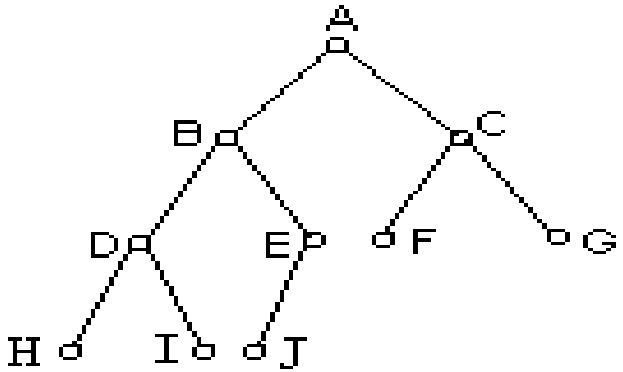
一般二叉树的顺序存储的问题

- 如果需要增加很多空结点才能将一棵一般的二叉树改造成完全二叉树，那么采用顺序表示就会造成空间的大量浪费，这时就不适合用顺序表示
- 对于一种特殊情况，如果二叉树为右单支树，其深度为k（有k+1个结点]，则需要一个长度为2^{k+1}-1 的一维数组，造成(2^{k+1}-1)-(k+1)个结点浪费。
  - ✓如果k=5, 则有57个结点空间浪费



完全二叉树的顺序表示

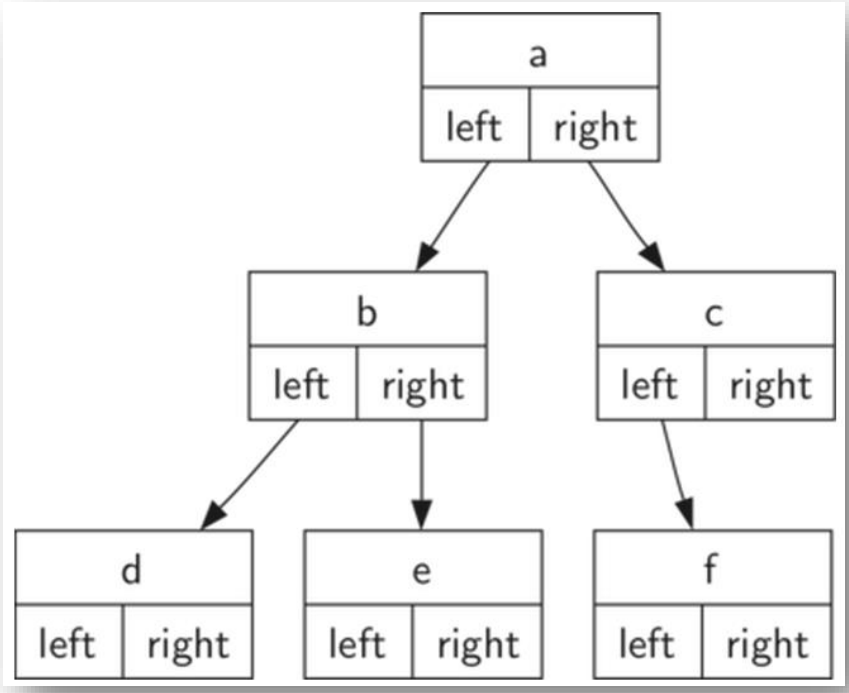
- 用一组连续的存储单元来存放二叉树中的结点
  - 完全二叉树中结点的层次序号可以反映出结点之间的逻辑关系
  - 例：完全二叉树的顺序表示（性质5）



	A	B	C	D	E	F	G	H	I	J
数组下标	0	1	2	3	4	5	6	7	8	9

二叉树的链式存储结构

- 同样可以用类似链表的节点链接法来实现树
  - –每个节点保存根节点的数据项，以及指向左右子树的链接
- 定义一个BinaryTree类
  - –成员key保存根节点数据项
  - –成员left/right_child则保存指向左/右子树的引用（同样是BinaryTree对象）




```
class BinaryTree:
    def __init__(self, rootobj):
        self.key = rootobj
        self.left_child = None
        self.right_child = None
```

二叉树的链式存储结构

- insert_left/right方法
  - 过程与线性表的节点插入相似

```
r = BinaryTree('a')
r.insert_left('b')
r.insert_right('c')
r.get_right_child().set_root_val('hello')
r.get_left_child().insert_right('d')
```

```
def insert_left(self, new_node):
    if self.left_child == None:
        self.left_child = BinaryTree(new_node)
    else:
        t = BinaryTree(new_node)
        t.left_child = self.left_child
        self.left_child = t

def insert_right(self, new_node):
    if self.right_child == None:
        self.right_child = BinaryTree(new_node)
    else:
        t = BinaryTree(new_node)
        t.right_child = self.right_child
        self.right_child = t

def get_right_child(self):
    return self.right_child

def get_left_child(self):
    return self.left_child

def set_root_val(self, obj):
    self.key = obj

def get_root_val(self):
    return self.key
```