第六节 树与二叉树

1. 课程介绍

* 1.什么是树;（了解）
* 2. 树的应用;（了解）
* 3.树的操作;（了解）
* 4. 什么是二叉树;(掌握)
* 5. 二叉树的性质(了解)
* 6.二叉树的实现(掌握)

1. 树的基本概念
   1. 什么是树?

**树(tree)**,是一种数据结构，它是由个结点组成一个具有层次关系的集合().

对任意一棵树,

* + - 1. 有且仅有一个特定的称为**根**(root)的结点;
      2. 当时,其余结点可分为个互不相交的有限集,且每个集合又是一棵树.称为在**子树**(subtree).

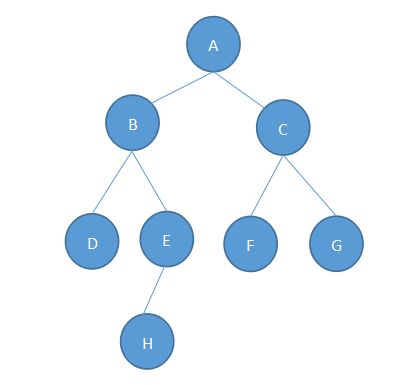


图1. 树结构.

* 1. 关于树的术语

**结点**:数据元素和若干指向子树的分支.

**结点的度**: **结点拥有的子树的数目**.

**子结点**(child): 结点的后继.

**双亲结点**(parent): 子结点的前驱.

**兄弟结点**(sibling):同一结点的子结点之间互称**兄弟**.

**祖先结点**:从根到该结点所经过的分支上的所有结点.

**子孙结点**: 以某结点为根的子树中任意一个结点.

**叶子**:**度为0的结点.(子树数为0的结点.)**

**分支结点**: 度不为0的结点.

**结点的层次**：根的层次定义为1，根的子结点层次为2，以此类推

**树:**

**树的度**: **树内各结点的度的最大值.**

**树的高度**：**树中结点的最大层次.**

**无序树**：树中结点的各子树的**次序不重要**，可以交换位置。如族谱.

**有序树**：树中结点的各子树的**次序重要**，不可交换位置.

**森林**：由m（m>=0）棵互不相交的树的集合.

森林与树之联系: 一棵树去掉根，其子树构成一个森林.

* 1. 树的特征

1. 每个结点有0个或多个子结点;

2. 根：无父结点的结点;

3. 除根外，每个结点只有一个父结点;

4. 每个子结点可分为多个不相交的子树.

* 1. 树的应用
     + 1. 二叉排序树: 每个结点数据中都有一个关键值,且有如下性质:对于每个结点,如果其左子树非空,则左子树的所有结点的关键值都小于该结点的关键值; 对于每个结点,如果其右子树非空,则右子树的所有结点的关键值都大于该结点的关键值.
       2. 哈夫曼树(最优树):一种带权重**路径长度最短**的树. 例如, 在实际应用中，各字符的出现频率是不相同的. 如A、B、C的使用频率远远高于X,Y,Z. 所以我们会想在设计编码时，让使用频率高的用短码，使用频率低的用长码，以优化整个信息的编码.
       3. html,xml.
       4. Ｍysql数据库索引.
       5. 文件系统的目录结构.
       6. 决策树 (机器学习)
       7. B树. 当大规模数据存储到磁盘中的时候，定位是一个很花费时间的过程. 我们可通过B树进行优化，提高磁盘读取时定位的效率.
  2. 树的基本操作(了解):

建树;

求树的根;

求树中结点的双亲结点;

求树中结点的第个子结点;

求树中结点的右兄弟;

将根为x在子树置为y结点的第个孩子;

删除x结点的第个孩子;

遍历;

置空树.

1. 二叉树
   1. 什么是二叉树?

**二叉树:**

递归定义:

1. 二叉树是个结点的有限集;
2. 当时,有且仅有一个结点为根结点,其余结点构成两个互不相交的子集;

又分别是一棵二叉树.

特点:

每个结点**最多有两棵子树**,且**子二叉树有左右之分**. 两棵子二叉树分别称为**左子树**和**右子树**.

**二叉树有5种形态**:

* + - 1. 空树
      2. 仅有根
      3. 右子树为空
      4. 左子树为空
      5. 左,右子树均在

**注意:**

1. **度为2的树**和**二叉树**是不同的概念: 在二叉树中,当一个结点的度为1时,其子树要考虑**序**.
2. **树不能为空,二叉树可以为空**.
   1. 二叉树的性质

**性质1**:**在二叉树的第层上的结点数最多为.**

证明: 满足;

假设第层上的结点数最多为;

每个结点最多有**两个**子结点,那么第层上的结点数最多为. 得证.

**性质2**:**在高度为的二叉树中至多有个结点().**

证明: 由性质1知,高度为k的二叉树最多结点为:

得证.[1]

具体做法:

层数I = 1, 结点最多为1

层数I = 2, 结点最多为2

层数I= 3, 结点数最多为4

….

层数I= k, 结点数为.

**性质3:若二叉树的叶子(度为0)结点数为,度为2的结点数为, 则:** .

证明: 二叉树中的结点的度只有0,1,2这三种可能值,设度为0结点个数为, 设度为1结点个数为, 设度为2结点个数为,则二叉树的总结点数为

又因二叉树的结点可分为**根结点**,度为1的结点的后续和度为2的结点的后续. 度为1的结点指向一个后续,度为2的结点指向2个后续,二叉树的总结点数可表示为:

,

解方程组,消去,得

.

得证. (根据两种分类方法,得到两个方程.)

**性质4**:**含个结点的二叉树的高度至少为.**

证明:设二叉树高度为由性质2得,

移项,得

,

对不等式两边同时取以2为底的对数,

.

得证.

* 1. 两种特殊形式的二叉树

**满二叉树**:

高度为，且有个结点的二叉树，称为**满二叉树**. 如下图, 高度 ,结点个数:

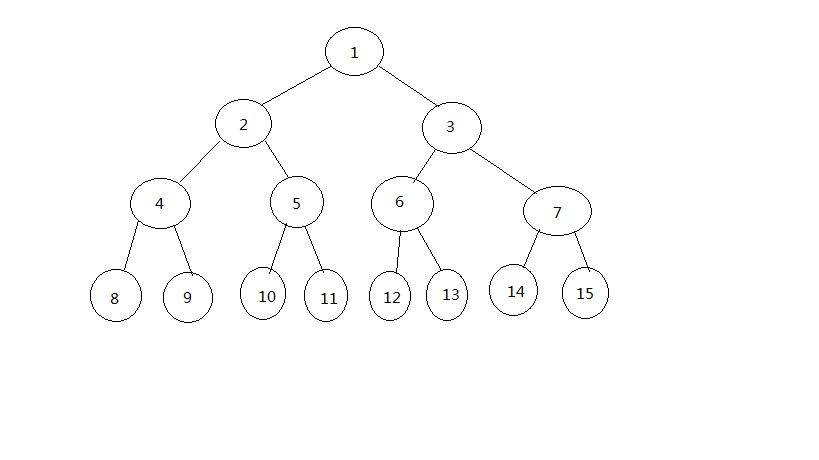


图2. 满二叉树.

**完全二叉树**:

一个深度为的二叉树,结点数为,且这个结点所构成的二叉树与深度为的满二叉树前 个结点位置相同,则该二叉树是一个深度为的**完全二叉树.**

**特征**: 深度为的完全二叉树, 其前层是满二叉树,第层的结点**依次排在左边**位置.

* 1. 二叉树的存储与表示

**顺序存储**

对于满二叉树或完全二叉树,可用**顺序存储方式**:用一个列表按结点标号顺序依次存放二叉树中的数据元素. 例如,对如图所示深度为3的满二叉树,其顺序存储结构如下:

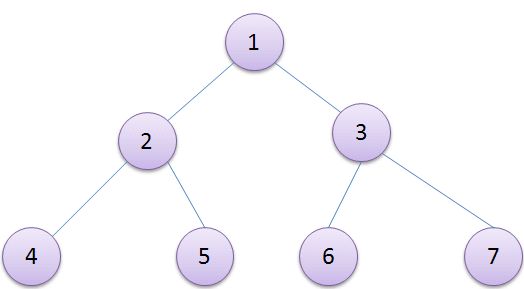


图3. 满二叉树可用顺序存储方式.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 下标 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 数据元素 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

顺序存储结构用得少.

**链式存储**

常用的链式结构有:二叉链表,三叉链表.

二叉链表:**每个结点有三个域,即数据域,左指针,右指针**.

三叉链表: 在二叉链表基础上,在结点中增加一个指向双亲结点的指针域.

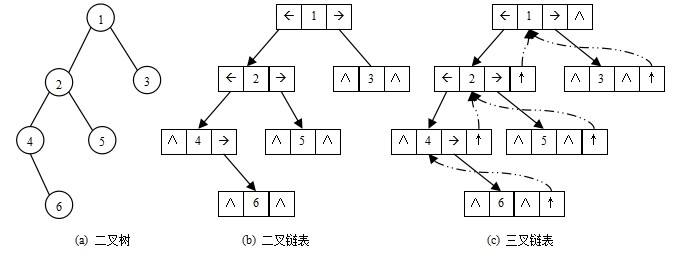


图4. 二叉树的链式存储方式. 其中, ”^”符号表示该指针指向NULL.

**二叉树的实现**

创建和添加结点

1. **class** Node(object):
2. """结点类"""
3. **def** \_\_init\_\_(self, elem=-1, lchild=None, rchild=None):
4. self.elem = elem
5. self.lchild = lchild
6. self.rchild = rchild
8. **class** Tree(object):
9. """树类"""
10. **def** \_\_init\_\_(self, root=None):
11. self.root = root
13. **def** add(self, elem):
14. """为树添加结点"""
15. node = Node(elem)
16. #如果树是空的，则对根结点赋值
17. **if** self.root == None:
18. self.root = node
19. **else**:
20. queue = []
21. queue.append(self.root)
22. #对已有的结点进行层次遍历
23. **while** queue:
24. #弹出队列的第一个元素
25. cur = queue.pop(0)
26. **if** cur.lchild == None:
27. cur.lchild = node
28. **return**
29. **elif** cur.rchild == None:
30. cur.rchild = node
31. **return**
32. **else**:
33. #如果左右子树都不为空，加入队列继续判断
34. queue.append(cur.lchild)
35. queue.append(cur.rchild)
    1. 二叉树的遍历

**按一定规律访问二叉树上的每个结点,且每个结点只能访问一次**，这种操作称为**遍历**（traversal）.

树的两种重要的遍历模式是**深度优先遍历**和**广度优先遍历**,深度优先一般用递归或栈，广度优先一般用队列.

* + 1. 深度优先遍历

对于一棵二叉树，深度优先搜索(Depth First Search,DFS)是沿着树的深度遍历其结点，尽可能深地搜索其分支.

二叉树由**三个基本单元**构成,即: 左子树, 右子树和根结点. 以L,R,T分别表示遍历左子树, 遍历右子树和访问根. 则有6种遍历方案: TLR, LTR, LRT, TRL, RTL, RLT. 二叉树的遍历通常指前三种: TLR, LTR, LRT. (LRT:先遍历左子树,后遍历右子树). 这三种遍历分别叫做前序遍历（preorder），中序遍历（inorder）和后序遍历（postorder）.

**前序遍历** (TLR)

先访问根，然后递归使用前序遍历访问左子树，再递归使用前序遍历访问右子树.

1. **def** preorder(self, root):
2. """递归实现前序遍历"""
3. **if** root == None:
4. **return**
5. **print**(root.elem)
6. self.preorder(root.lchild)
7. self.preorder(root.rchild)

**中序遍历** (LTR)

先递归使用中序遍历访问左子树，然后访问根，最后再递归使用中序遍历访问右子树.

1. **def** inorder(self, root):
2. """递归实现中序遍历"""
3. **if** root == None:
4. **return**
5. self.inorder(root.lchild)
6. **print**(root.elem)
7. self.inorder(root.rchild)

**后序遍历** (LRT)

先递归使用后序遍历分别访问左子树和右子树，最后访问根结点.

1. **def** postorder(self, root):
2. """递归实现后续遍历"""
3. **if** root == None:
4. **return**
5. self.postorder(root.lchild)
6. self.postorder(root.rchild)
7. **print**(root.elem)
   * 1. 广度优先遍历(层次遍历)

层次遍历,是通过**队列**实现. 从根开始，从上到下,从左到右遍历整个树的结点.

1. **def** breadth\_travel(self, root):
2. """利用队列实现树的层次遍历"""
3. **if** root == None:
4. **return**
5. queue = []
6. queue.append(root)
7. **while** queue:
8. node = queue.pop(0)
9. **print**(node.elem)
10. **if** node.lchild != None:
11. queue.append(node.lchild)
12. **if** node.rchild != None:
13. queue.append(node.rchild)

注:

Python3里面已有一个名为queue的模块. 里面包含了队列,栈和堆的基本操作. [https://docs.python.org/3/library/queue.html#module-queue](https://docs.python.org/3/library/queue.html" \l "module-queue)

1. 课程总结
   1. 重点
      * 1. 掌握树的基本概念和相关术语
        2. 二叉树的基本概念和性质
   2. 难点
      * 1. 理解二叉树的性质
        2. 二叉树的遍历的实现
2. 课后练习
   * + 1. 判断题: 在二叉树中,当一个结点的度为1时,其子二叉树要考虑序.
3. 正确
4. 错误
   * + 1. 按照二叉树的定义,具有3个结点的二叉树有\_\_\_\_种.
5. 3
6. 4
7. 5
8. 6
   * + 1. 画出图5所示的二叉树的二叉链表存储结构.

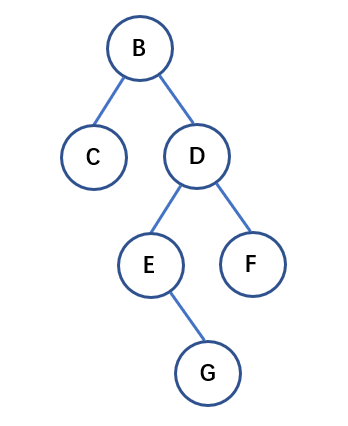


图5.

* + - 1. 图5所示的二叉树的前序遍历序列为\_\_\_\_\_\_\_\_\_\_\_\_, 中序遍历序列为\_\_\_\_\_\_\_\_\_\_\_\_,后序遍历序列为\_\_\_\_\_\_\_\_\_\_\_\_\_.

1. 注释

[1] 等比数列 (为首项,为公比)前项和的公式: .

1. 扩展知识或课外阅读推荐
   1. 课外阅读

刘清,张小勇,王琼 <<数据结构>>(第三版),电子工业出版社 (2010)

裘宗燕,<<数据结构与算法>>,机械工业出版社 (2015)