终于有机会重新回头学习一下一直困扰自身多年的数据结构了，赶脚棒棒哒。一直以来，对数据结构的掌握基本局限于线性表，稍微对树有一丢丢了解，而对于图那基本上就是不懂（不可否认，很多的考试中回避了图也是原因之一），而查找和排序只能算是了解点皮毛，简单的面试能应付的水平。关于数据结构方面的教材和视频有不少，首推严蔚敏老教授的书和视频，尤其是视频，记载的是其在清华大学的授课过程，全程通过不同的教具来演示不同的示例，非常直观。自身由于懒惰，一直也没坚持的把其看完，于是选择了相对简单的学习方法，就是选择了程杰老师的《大话数据结构》一书，该数诙谐幽默，让人的挫败感下降了很多，老话说的好，“兴趣是一切学习的老师”，别小看任何入门书，它往往是决定你坚持你努力的初衷。随着年龄的增长，会发现一个新的开始是越来越难，推荐一篇园内博文《学习新东西的唯一方法》<http://kb.cnblogs.com/page/536332/>。

言归正传，本文主要介绍数据结构中的树，可以说树是数据结构最为承上启下的部分，其可以转化为线性表（通过**二叉树的线索化**），也是学习图的基础，此外在实际中，数据库索引使用的**B+树**数据结构也是经典中的经典。

树的定义：是n个结点的有限集，在任何非空树中，有且仅有一个**根结点**，其余结点可以分为m个互不相交的有限集，其中的内一个集合又是一棵树，并称为根的**子树**。一种简单的描述为，树是由一个根节点和若干子树构成，树中结点具有相同数据类型和层次关系，是一种类似链表的递归式结构。

树中结点的定义：树的结点包含一个数据元素及若干指向其子树的分支，结点拥有的子树数称为结点的**度**(Degree)。度为0的结点称为叶子结点，大于0的为分支结点，树的度是树内各结点度的最大值。结点的子树称为该节点的**孩子**，相应的该节点称为孩子的**双亲**(Parent)。同一个双亲的孩子之间互称兄弟(Sibling)，结点的祖先是从根到该节点所经分支上的所有节点。

此外，森林是m棵互不相交的树的集合，对于任意一个结点来说，其子树的集合即为森林。



图 1树的基本概念

了解了树的基本概念，接下来学习树的存储结构，常见的包括**双亲表示法、孩子表示法和孩子兄弟表示法**。存储结构的设计是一个非常灵活的过程，一个存储结构设计的是否合适，取决于基于该存储结构的运算是否合适、是否方便、时间复杂度好不好。不过一般来说，一个树结点都具有**数据域**和**指针域**，之后将分别介绍之前提及的三种表示法。

**双亲表示法**，相对比较简单，之后的例子为了方便兄弟的查找还添加了右孩子指针。

|  |
| --- |
| public class Tree1  {  public TreeNode[] NodeList { get; set; }  public int NodeCount { get; set; }  public int RootIndex { get; set; }  private class TreeNode  {  public object Data { get; set; }  public TreeNode Parent { get; set; }  public TreeNode RightSibling { get; set; }  }  } |

**孩子表示法**，由于每个结点可以有多个孩子，因此可以使用多重链表的表示法，在结点中，使用多个指针，每个指向一个孩子。此时有两种解决方案来构建器存储结构，一种是指针的个数就是树的度，缺点是由于每个结点的度和树的度可能差异很大，会造成大量的空间浪费；另一种是添加一个**度域**，然后根据结点的度来构建指针域，其缺点是每个结点的结构不同，造运算性能上的下降。那有没有更好的方法，即保证结点结构相同并且避免空指针带来的空间浪费。必须是有的嘛，那就是将每个结点的孩子结点排列起来，以单链表作存储结构，n个节点构建n个孩子链表，叶子结点只有头指针。然后将n个头指针组成一个线性表，采用顺序结构存放在一位数组中。因此需要设计**头结点**和**链表的孩子结点**两种结构，整体方案如下所示。



图 2树的孩子表示法

|  |
| --- |
| public class Tree2  {  public TreeNode[] NodeList { get; set; }  public int NodeCount { get; set; }  public int RootIndex { get; set; }  public class TreeNode  {  public object Data { get; set; }  public TreeNode FirstChild { get; set; }  }  public class LinkedNode  {  public int CurIndex { get; set; }  public TreeNode Next { get; set; }  }  } |

**孩子双亲表示法**，通过对之前树进行观察，不难发现，一个节点的第一个孩子如果存在就是唯一的，有兄弟也是如此，因而该表示法由数据域、firstchild指针和rightSibling指针组成，由于和之前类似，省略代码，这里的重点是，通过这种表示法，你会发现该表示法其实就是将该树转化为了一个特殊的树**（二叉树，树结构中的核心）**，一个值域，两个指向左右孩子的指针域，在下一节中将进行详细介绍。

* 二叉树

**二叉树的定义**：二叉树（Binary Tree）是一个特殊的树，由一个根结点和两个互不相交的、分别被称为左子树和右子树的二叉树组成。仍然是一个递归的概念，在与树有关的结构中，有很多往往最后都是通过转化为二叉树，再借助二叉树相关算法，达到化繁为简的效果。为了更好的学习二叉树，之后将介绍几种特殊的树。

斜树：所有的结点只有左或右子树的二叉树。

满二叉树：所有分支结点都存在左子树和右子树，且所有叶子都在同一层上的二叉树。

**完全二叉树**：最重要的一种特殊二叉树，理解起来也有一定难度。定义是，对一颗具有n个结点的二叉树按层序编号，如果编号为i的结点与相同深度的满二叉树中编号为i的结点在二叉树上的位置完全相同，则称其为完全二叉树，相关的图例如下。



图 3特殊的二叉树

可以发现完全二叉树特点：叶子结点只能出现在最下两层；最下层叶子一定集中在左侧连续位置；倒数第二层叶子结点一定在右侧连续位置；如果结点度为1，也该结点只有左孩子；同样结点数的二叉树，完全二叉树深度最小。

**二叉树的性质**

1．在二叉树的第i层上至多有2i-1个结点。

2．深度为k的二叉树至多有2i-1个结点。

3．对任意一颗二叉树，如果其终端结点数为n0，度为2的节点数为n2，则**n0=n2+1**。终端结点也就是叶子结点，除此之外就是度为1或2的结点数了，有结点总数**n=n0+n1+n2**（式1）。通过连接线的角度来看，由于根结点只有分支出去，没有分支进入，因此连接线数为m = n-1；再从度的角度出发，有连接数m = n1+2n2，因而推导出**n-1= n1+2n2**（式2）,联立式1、2得**n0=n2+1**。

4．具有n个结点的完全二叉树的深度为⌊log2n + 1⌋，符号为向下取整。

5.如果对一个n个结点完全二叉树按层序编号，对任意结点i有：如果i=1，则结点i是二叉树的根，如果i>1，则其双亲是结点⌊i/2⌋；如果2i>n，则结点i无左孩子，否则其左孩子为结点2i；如果2i+1>n，则结点i无右孩子，否则其右孩子为2i+1。

Tip:

Word快捷操作, 指数——按[组合键](https://www.baidu.com/s?wd=%E7%BB%84%E5%90%88%E9%94%AE&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBnvP-uj7Wrj-9uhwWnWRv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPHRLnHRvns)“Ctrl+Shift+ =”上标，重复操作则恢复常规输入；下标—— 按[组合键](https://www.baidu.com/s?wd=%E7%BB%84%E5%90%88%E9%94%AE&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBnvP-uj7Wrj-9uhwWnWRv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPHRLnHRvns)“Ctrl+ =”下标，重复操作则恢复常规输入。

**二叉树的存储结构**：包括顺序存储和二叉链表存储两种方式。对于任意的二叉树，我们把它模拟成一个对应的完全二叉树，其按照层序编号，可以很方便的存放在一维数组中，请见以下示例。



图 4二叉树的顺序存储结构

其中^表示不存在的结点，非常的方便的就将一个树的存储简化为了线性表的存储。顺序存储的缺点是，如果该二叉树是一个右斜树，那么会造成存储空间很大的浪费，此时需要考虑使用二叉链表的存储结构。这种结构理解起来很简单，即结点包含一个数据域和左右孩子两个指针域。

**遍历二叉树**：二叉树的遍历(traveling binary tree)的定义为指从根结点出发，按照**某种次序**依次访问二叉树中所有的结点，且每个结点被有且仅有一次访问。这个定义的重点是次序，其引申出二叉树遍历的4种方式：前序遍历、中序遍历、后序遍历和层序遍历，这儿的前中后均表示访问父结点的次序，示例图如下所示。



图 5二叉树的遍历方法

经过观察，无论是那种遍历方式，都将树这个复杂的逻辑结构转化为了简单的线性结构，称为了计算机真正可以处理的数据结构。接下来更进一步，通过代码来描述遍历方法，由于几种方式形式相近，这儿就选取中序遍历为例。

|  |
| --- |
| public class BinaryTree  {  public TreeNode Root { get; set; }  public void MiddleOrderTravel() { TreeManager.MiddleOrderTravel(Root); }  }  public class TreeNode  {  public object data { get; set; }  public TreeNode LeftChild { get; set; }  public TreeNode RightChild { get; set; }  }  public class TreeManager  {  public static void MiddleOrderTravel(TreeNode tree)  {  if (null == tree) return;  MiddleOrderTravel(tree.LeftChild);  Console.WriteLine(tree.data);  MiddleOrderTravel(tree.RightChild);  }  } |

接下里介绍一个很常见的数据结构考题，用于熟悉二叉树遍历的相关概念。一颗二叉树的前序遍历为ABDCFG，中序遍历为DBAFCG，那么这颗树后续遍历的结果是什么?其结构是怎么样的？分析过程如下所示。

通过前序遍历的ABDCFG，可以该树的根结点为A；再通过中序遍历可知A的左子树为DB，右子树为FCG；分析左子树DB，从前序遍历ABD可以看出B是该子树的根，通过中序遍历DBA可知，D为B的左孩子；对于右子树FCG，通过前序遍历ABDC可知，C为该子树的根，再通过中序遍历FCG可知，F为左子树，G为右子树；即为之前图5中的二叉树，后续遍历为DBFGCA。

此外，需要知道的是，已知前序遍历和中序遍历、后序遍历和中序遍历可以唯一确认一个二叉树，而前序遍历和后序遍历不能。例如前序遍历为ABC，后序遍历为CBA，该树的中序遍历可以为BAC,也可以是CBA等。

* 线索二叉树

之前介绍通过二叉链表的方式来存储二叉树，可以比较好的利用空间，但仍然有很多的空指针域存在（对于n个结点的二叉树，其指针域数量为2n，分支数为n-1，其空指针域有2n – (n-1) = n + 1个），造成空间的浪费，那么有木有什么好的办法将其利用起来呢？

那么就是空指针域记录结点的前驱和后继（和线性表一样），也称这些指针为线索，加上线索的二叉链表称为线索链表，相应的二叉树为线索二叉树(Threaded Binary Tree)，以下通过以中序遍历为例，构建一个线索二叉树。



图 6二叉树的线索化

上图比较简陋，望见谅。简单来说，就是将结点空闲的左孩子指针记录其前驱，右孩子指针记录其后继，添加一个头结点指针方便遍历。此外，需要为结点增加左、右两个标识域，用于记录其指针是指向的其孩子还是其前驱/后继，线索二叉树的结构和创建代码如下所示。

|  |
| --- |
| public class ThreadedBinaryTreeNode  {  public ThreadedBinaryTreeNode Root { get; set; }  public void MiddleOrderTravel() { ThreadedBinaryManager.InThreading(Root); }  }  public class ThreadedBinaryTreeNode  {  public object data { get; set; }  public ThreadedBinaryTreeNode LeftChild { get; set; }  public ThreadedBinaryTreeNode RightChild { get; set; }  public TagType LeftTag { get; set; }  public TagType RightTag { get; set; }  }  public enum TagType : short  {  Child = 0,  Thread = 1  }  public class ThreadedBinaryManager  {  //用于记录前驱结点  public static ThreadedBinaryTreeNode PreNode { get; set; }  //中序遍历场景  public static void InThreading(ThreadedBinaryTreeNode tree)  {  if (null == tree) return;  InThreading(tree.LeftChild);  if (null == tree.LeftChild)  {  tree.LeftTag = TagType.Thread; //前驱线索  tree.LeftChild = PreNode; //指向前驱  }  if (null == tree.RightChild)  {  tree.RightTag = TagType.Thread; //后继线索  tree.RightChild = tree; //指向后继，即当前结点p  }  PreNode = tree;  InThreading(tree.RightChild);  }  } |

* 树、森林与二叉树的转换

这部分内容有一些繁杂，需要耐心理解，其主要意图还是通过将树、森林转化为二叉树来简化问题，包括：树转化为二叉树，森林转化为二叉树，二叉树转化为树，二叉树转换为森林和树与森林的遍历。

**树转化为二叉树**，包含3个步骤：**加线**，在所有兄弟结点之间加一条连线；**去线**，对树中每个结点，只保留它与第一个孩子结点的连线，三处它与其他孩子结点之间的连线；**层次调整**，以树的根结点为轴心，将整棵树顺时针旋转一定角度，使其层次分明。相关示例如下所示。



这儿需要提及的一点是，转换后的二叉树的左子树为其第一个孩子，右子树为其兄弟，虽然以上的转化结果看起来怪怪的，但确实是这样的结果。

**森林转化为二叉树**

* 赫夫曼树及其应用
* B树
* B+树

参考资料:

1. 程杰. 大话数据结构[M]. 北京:清华大学出版社, 2011.
2. 严蔚敏, 吴伟民. 数据结构（C语言版）[M]. 北京:清华大学出版社, 2004.