数据结构

Tedu Python 教学部
Author：吕泽
Days：2天

数据结构基本概念

什么是数据结构？

1. 数据

> 数据即信息的载体，是能够输入到计算机中并且能被计算机识别、存储和处理的符号总称。

2. 数据元素

> 数据元素是数据的基本单位，又称之为记录（Record）。一般，数据元素由若干基本项（或称字段、域、属性）组成。

3. 数据结构

> 数据结构指的是数据元素及数据元素之间的相互关系，或组织数据的形式。

数据之间的结构关系

1. 逻辑结构

> 表示数据之间的抽象关系（如邻接关系、从属关系等），按每个元素可能具有的直接前趋数和直接后继数将逻辑结构分为“线性结构”和“非线性结构”两大类。

2. 存储结构

> 逻辑结构在计算机中的具体实现方法，分为顺序存储方法、链接存储方法、索引存储方法、散列存储方法。

逻辑结构（关系）

1. 特点：

- 只是描述数据结构中数据元素之间的联系规律
- 是从具体问题中抽象出来的数学模型，是独立于计算机存储器的（与机器无关）

2. 逻辑结构分类

- 线性结构

> 对于数据结构课程而言，简单地说，线性结构是n个数据元素的有序（次序）集合。
>
> > - 集合中必存在唯一的一个"第一个元素"；
> > - 集合中必存在唯一的一个"最后的元素"；
> > - 除最后元素之外，其它数据元素均有唯一的"后继"；
> > - 除第一元素之外，其它数据元素均有唯一的"前驱"。

- 树形结构（层次结构）

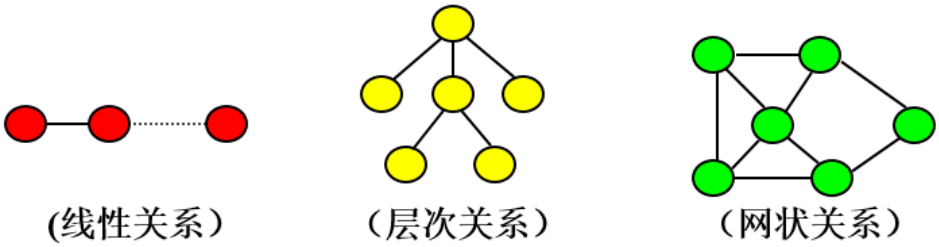
> 树形结构指的是数据元素之间存在着“一对多”的树形关系的数据结构，是一类重要的非线性数据结构。在树形结构中，树根结点没有前驱结点，其余每个结点有且只有一个前驱结点。叶子结点没有后续结点，其余每个结点的后续节点数可以是一个也可以是多个。

- 图状结构（网状结构）

> 图是一种比较复杂的数据结构。在图结构中任意两个元素之间都可能有关系，也就是说这是一种多对多的关系。

- 其他结构

> 除了以上几种常见的逻辑结构外，数据结构中还包含其他的结构，比如集合等。有时根据实际情况抽象的模型不止是简单的某一种，也可能拥有更多的特征。



存储结构（关系）

1. 特点：

- 是数据的逻辑结构在计算机存储器中的映象（或表示）
- 存储结构是通过计算机程序来实现的，因而是依赖于具体的计算机语言的。

2. 存储结构分类

- 顺序存储

> 顺序存储（Sequential Storage）：将数据结构中各元素按照其逻辑顺序存放于存储器一片连续的存储空间中。

- 链式存储

> 链式存储（Linked Storage）：将数据结构中各元素分布到存储器的不同点，用记录下一个结点位置的方式建立它们之间的联系，由此得到的存储结构为链式存储结构。

- 索引存储

> 索引存储（Indexed Storage）：在存储数据的同时，建立一个附加的索引表，即索引存储结构=数据文件+索引表。

- 散列存储

> 散列存储(Hash Structure)： 根据数据元素的特殊字段(称为关键字key)，计算数据元素的存放地址，然后数据元素按地址存放，所得到的存储结构为散列存储结构(或Hash结构)。

线性表

线性表的定义是描述其逻辑结构，而通常会在线性表上进行的查找、插入、删除等操作。
线性表作为一种基本的数据结构类型，在计算机存储器中的映象（或表示）一般有两种形式，一种是顺序映象，一种是链式映象。

线性表的顺序存储

1. 定义

> 若将线性表L=(a0,a1, ……,an-1)中的各元素依次存储于计算机一片连续的存储空间，这种机制表示为线性表的顺序存储结构。

2. 特点

> - 逻辑上相邻的元素 ai, ai+1，其存储位置也是相邻的；
> - 存储密度高，方便对数据的遍历查找。
> - 对表的插入和删除等运算的效率较差。

3. 程序实现

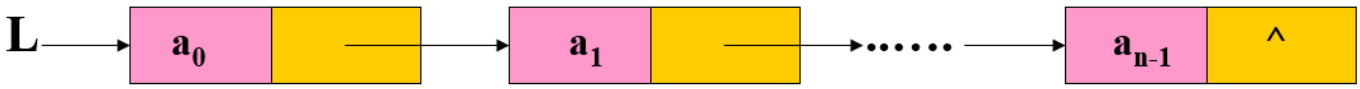
> 在Python中，list存放于一片单一连续的内存块，故可借助于列表类型来描述线性表的顺序存储结构，而且列表本身就提供了丰富的接口满足这种数据结构的运算。

```
>>>L = [1,2,3,4]
>>>L.append(10)       #尾部增加元素
L
[1, 2, 3, 4, 10]

>>>L.insert(1,20)     #插入元素
L
[1, 20, 2, 3, 4, 10]

>>>L.remove(3)        #删除元素
L
[1, 20, 2, 4, 10]

>>>L[4] = 30          #修改
L
[1, 20, 2, 4, 30]

>>>L.index(2)         #查找
2
```

线性表的链式存储

1. 定义

> 将线性表L=(a0,a1,……,an-1)中各元素分布在存储器的不同存储块，称为结点，每个结点（尾节点除外）中都持有一个指向下一个节点的引用，这样所得到的存储结构为链表结构。



2. 特点

> - 逻辑上相邻的元素 ai, ai+1，其存储位置也不一定相邻；
> - 存储稀疏，不必开辟整块存储空间。
> - 对表的插入和删除等运算的效率较高。
> - 逻辑结构复杂，不利于遍历。

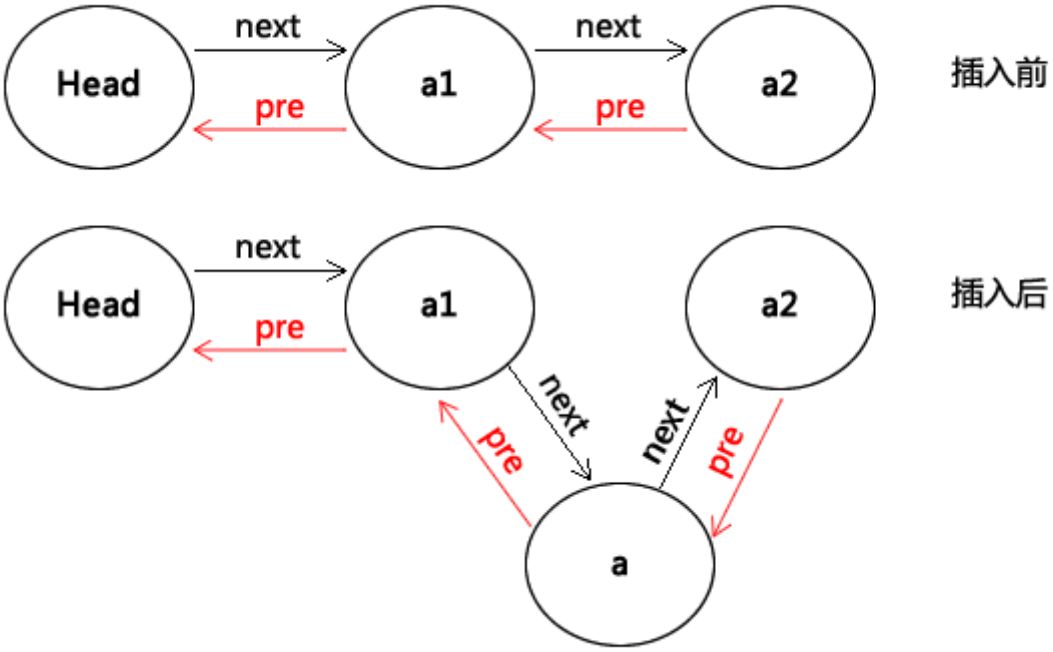
3. 程序实现

代码实现： day2/linklist.py

@ 扩展延伸：双向循环链表

1. 双向链表结点 ：

2. 双向循环链表实现：



代码实现：day2/dc_linklist.py

栈和队列

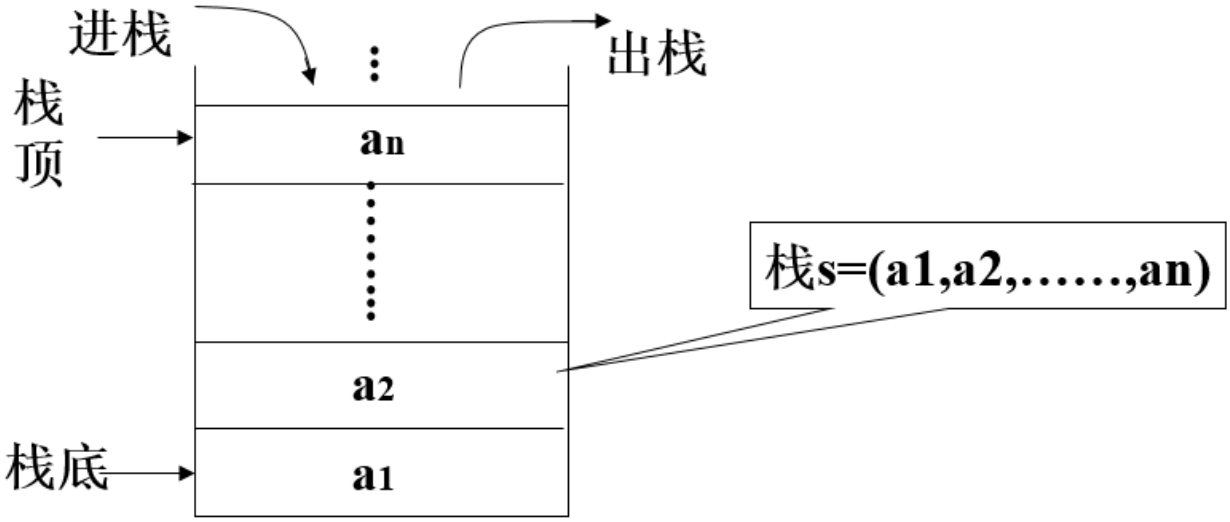
栈

1. 定义

> 栈是限制在一端进行插入操作和删除操作的线性表（俗称堆栈），允许进行操作的一端称为“栈顶”，另一固定端称为“栈底”，当栈中没有元素时称为“空栈”。

2. 特点：

> - 栈只能在一端进行数据操作
> - 栈模型具有后进先出或者叫做后进先出的规律



3. 栈的代码实现

栈的操作有入栈（压栈），出栈（弹栈），判断栈的空满等操作。

顺序存储代码实现： day2/sstack.py
链式存储代码实现： day2/lstack.py

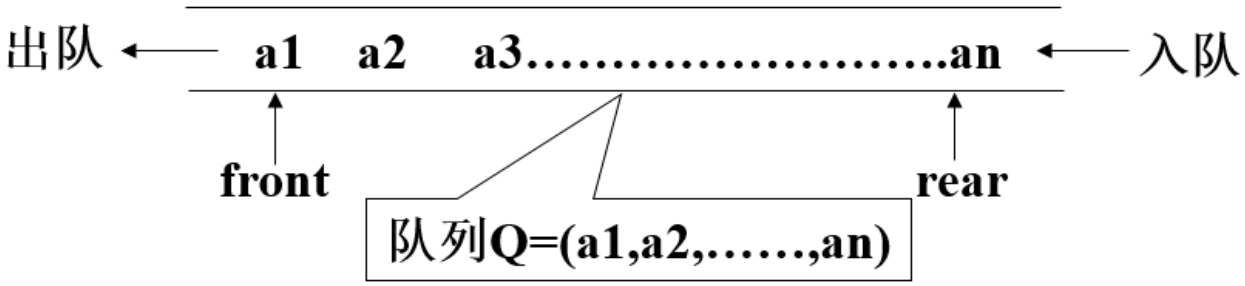
队列

1. 定义

> 队列是限制在两端进行插入操作和删除操作的线性表，允许进行存入操作的一端称为“队尾”，允许进行删除操作的一端称为“队头”。

2. 特点：

> - 队列只能在队头和队尾进行数据操作
> - 栈模型具有先进先出或者叫做后进后出的规律



3. 队列的代码实现

队列的操作有入队，出队，判断队列的空满等操作。

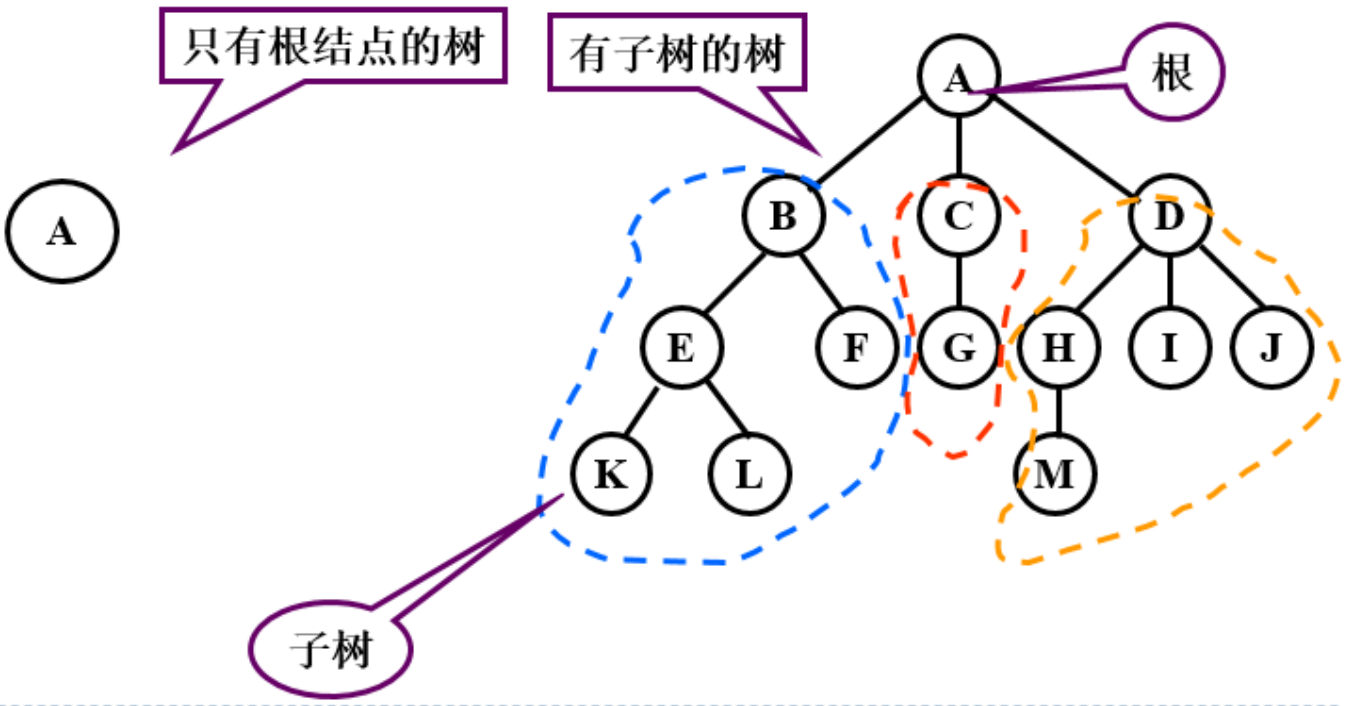
顺序存储代码实现： day3/squeue.py
链式存储代码实现： day3/lqueue.py

树形结构

基础概念

1. 定义

> 树（Tree）是n（n≥0）个节点的有限集合T，它满足两个条件：有且仅有一个特定的称为根（Root）的节点；其余的节点可以分为m（m≥0）个互不相交的有限集合T1、T2、……、Tm，其中每一个集合又是一棵树，并称为其根的子树（Subtree）。

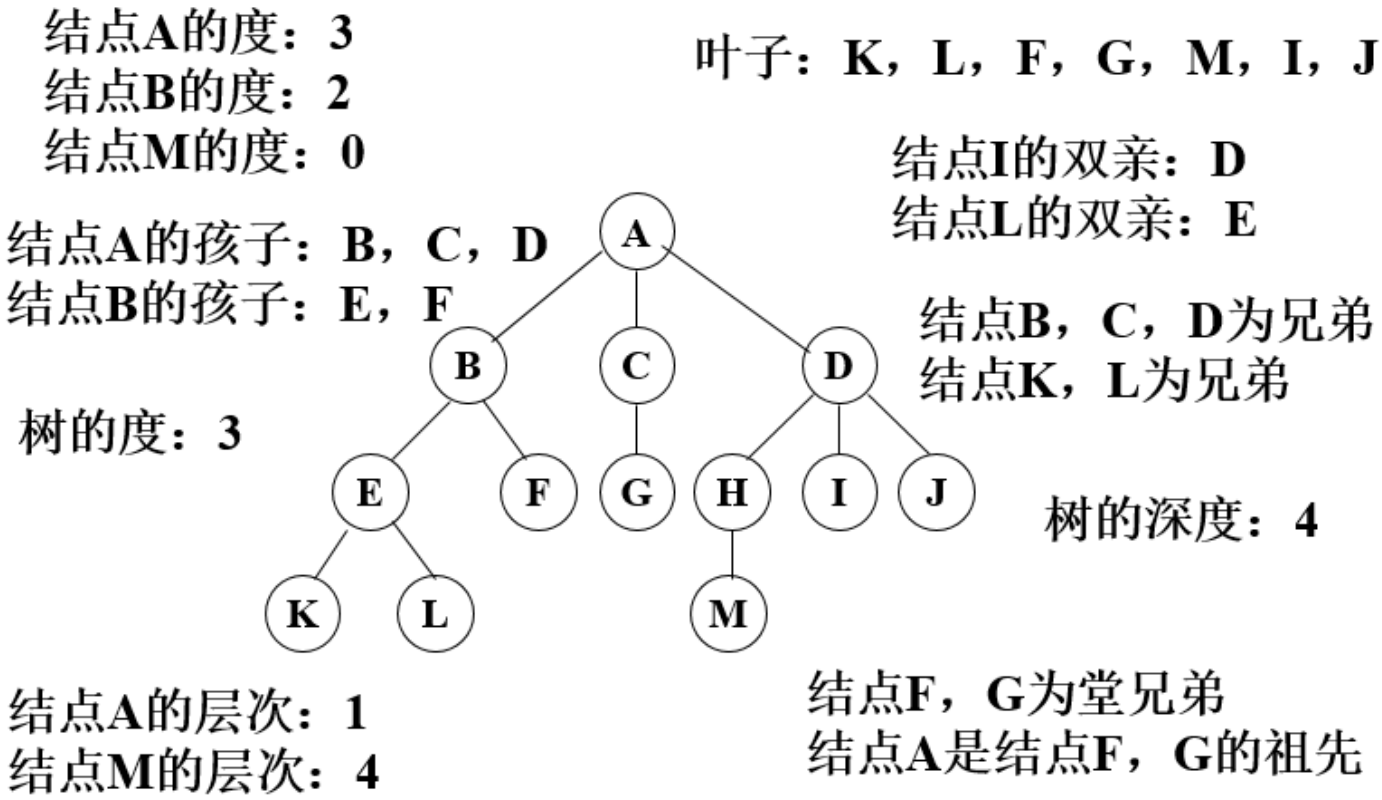


2. 基本概念

- 一个节点的子树的个数称为该节点的度数，一棵树的度数是指该树中节点的最大度数。
- 度数为零的节点称为树叶或终端节点，度数不为零的节点称为分支节点，除根节点外的分支节点称为内部节点。
- 一个节点的子树之根节点称为该节点的子节点，该节点称为它们的父节点，同一节点的各个子节点之间称为兄弟节点。一棵树的根节点没有父节点，叶节点没有子节点。
- 一个节点系列k1,k2, ……,ki,ki+1, ……,kj,并满足ki是ki+1的父节点，就称为一条从k1到kj的路径，路径的长度为j-1,即路径中的边数。路径中前面的节点是后面节点的祖先，后面节点是前面节点的子孙。

> - 节点的层数等于父节点的层数加一，根节点的层数定义为一。树中节点层数的最大值称为该树的高度或深度。

> - m（m≥0）棵互不相交的树的集合称为森林。树去掉根节点就成为森林，森林加上一个新的根节点就成为树。

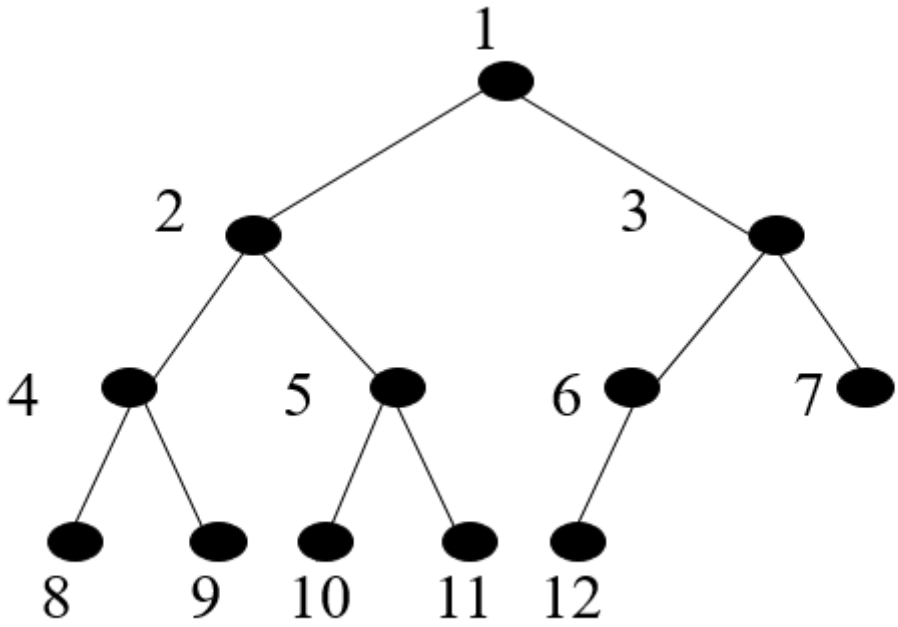


二叉树

定义与特征

1. 定义

> 二叉树（Binary Tree）是n（n≥0）个节点的有限集合，它或者是空集（n = 0），或者是由一个根节点以及两棵互不相交的、分别称为左子树和右子树的二叉树组成。二叉树与普通有序树不同，二叉树严格区分左孩子和右孩子，即使只有一个子节点也要区分左右。



2. 二叉树的特征

- 二叉树第i（i≥1）层上的节点最多为2^{i-1}个。
- 深度为k（k≥1）的二叉树最多有2^k-1个节点。
- 在任意一棵二叉树中，树叶的数目比度数为2的节点的数目多一。
- 满二叉树 ：深度为k（k≥1）时有2k - 1个节点的二叉树。
- 完全二叉树 ：只有最下面两层有度数小于2的节点，且最下面一层的叶节点集中在最左边的若干位置上。

二叉树的遍历

> 遍历 ：沿某条搜索路径周游二叉树，对树中的每一个节点访问一次且仅访问一次。

> 先序遍历： 先访问树根，再访问左子树，最后访问右子树；
> 中序遍历： 先访问左子树，再访问树根，最后访问右子树；
> 后序遍历： 先访问左子树，再访问右子树，最后访问树根；
> 层次遍历: 从根节点开始，逐层从左向右进行遍历。

递归思想和实践

1. 什么是递归？

所谓递归函数是指一个函数的函数体中直接调用或间接调用了该函数自身的函数。这里的直接调用是指一个函数的函数体中含有调用自身的语句，间接调用是指一个函数在函数体里有调用了其它函数，而其它函数又反过来调用了该函数的情况。

2. 递归函数调用的执行过程分为两个阶段

> 递推阶段：从原问题出发，按递归公式递推从未知到已知，最终达到递归终止条件。
> 回归阶段：按递归终止条件求出结果，逆向逐步代入递归公式，回归到原问题求解。

3. 优点与缺点

> 优点：递归可以把问题简单化，让思路更为清晰,代码更简洁
> 缺点：递归因系统环境影响大，当递归深度太大时，可能会得到不可预知的结果

递归示例： day2/recursion.py

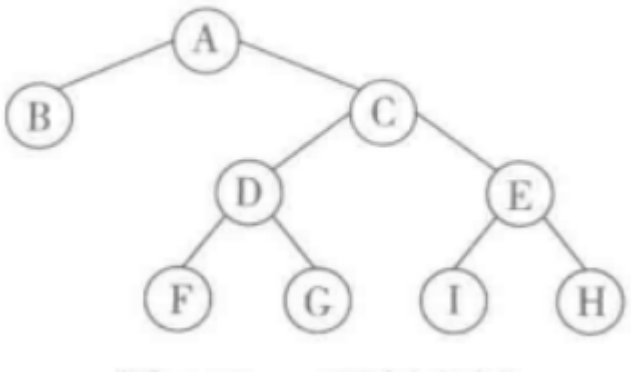
二叉树的代码实现

二叉树顺序存储

二叉树本身是一种递归结构，可以使用Python list 进行存储。但是如果二叉树的结构比较稀疏的话浪费的空间是比较多的。

- 空结点用None表示
- 非空二叉树用包含三个元素的列表[d,l,r]表示，其中d表示根结点，l，r左子树和右子树。

```
['A',['B',None,None
     ],
     ['C',['D',['F',None,None],
               ['G',None,None],
          ],
          ['E',['H',None,None],
               ['I',None,None],
          ],
     ]
]
```



二叉树链式存储

二叉树遍历： day2/bitree.py

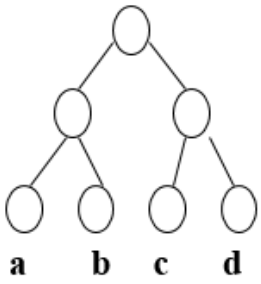
@ 扩展延伸：哈夫曼树

赫夫曼(Huffman)树，又称最优树，是带权路径长度最短的树，有着广泛的应用

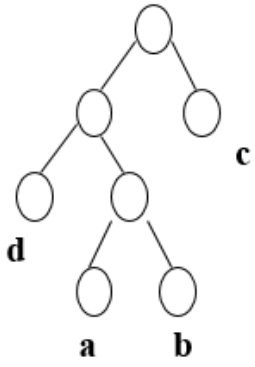
从树中一个结点到另外一个结点的分支构成一条路径，分支的数目称为路径的长度。树的路径长度是指从树根到每个结点的路径长度之和

进一步推广，考虑带权的结点。结点的带权路径长度指的是从树根到该结点的路径长度和结点上权的乘积。树的带权路径长度是指所有叶子节点的带权路径长度之和，记作 WPL 。WPL最小的二叉树就是最优二叉树，又称为赫夫曼树

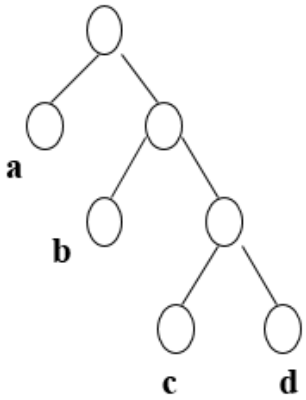
例如下面的三棵二叉树，都有四个叶子结点a，b，c，d，权值分别为7，5，2，4。



WPL = 7*2+5*2+2*2+4*2



WPL = 7*3+5*3+2*1+4*2



WPL = 7*1+5*2+2*3+4*3

思考: 如何构建一个哈夫曼树？

算法基础

基础概念特征

1. 定义

> 算法（Algorithm）是一个有穷规则（或语句、指令）的有序集合。它确定了解决某一问题的一个运算序列。对于问题的初始输入，通过算法有限步的运行，产生一个或多个输出。

数据的逻辑结构与存储结构密切相关:

- 算法设计: 取决于选定的逻辑结构
- 算法实现: 依赖于采用的存储结构

2. 算法的特性

- 有穷性 —— 算法执行的步骤（或规则）是有限的；

- 确定性 —— 每个计算步骤无二义性；
- 可行性 —— 每个计算步骤能够在有限的时间内完成；
- 输入 ，输出 —— 存在数据的输入和出输出

3. 评价算法好坏的方法

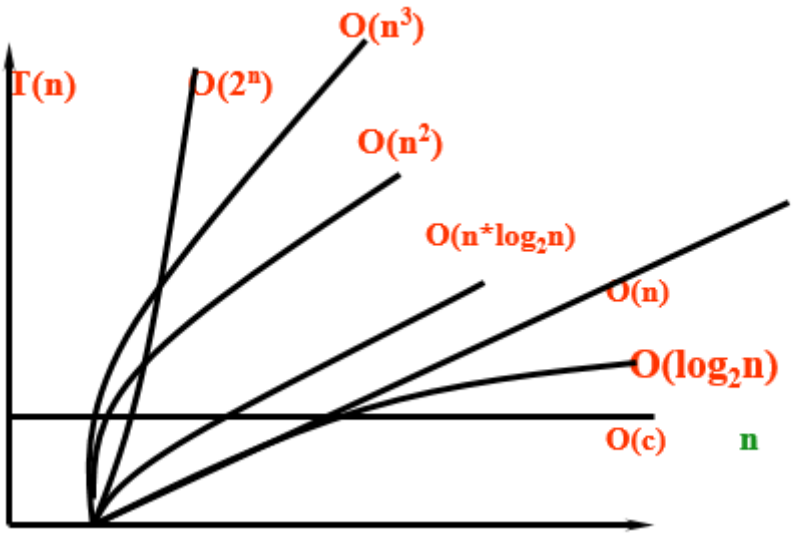
- 正确性：运行正确是一个算法的前提。
- 可读性：容易理解、容易编程和调试、容易维护。
- 健壮性：考虑情况全面，不容以出现运行错误。
- 时间效率高：算法消耗的时间少。
- 储存量低：占用较少的存储空间。

时间复杂度计算

算法效率——用依据该算法编制的程序在计算机上执行所消耗的时间来度量。“O”表示一个数量级的概念。根据算法中语句执行的最大次数（频度）来 估算一个算法执行时间的数量级。

> 计算方法：
>
> > 写出程序中所有运算语句执行的次数，进行加和
> > 如果得到的结果是常量则时间复杂度为1
> > 如果得到的结果中存在变量n则取n的最高次幂作为时间复杂度

下图表示随问题规模n的增大，算法执行时间的增长率。



排序和查找

排序

排序(Sort)是将无序的记录序列（或称文件）调整成有序的序列。

常见排序方法：

- 冒泡排序

> 冒泡排序是一种简单的排序算法。它重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。

- 选择排序

> 工作原理为，首先在未排序序列中找到最小元素，存放到排序序列的起始位置，然后，再从剩余未排序元素中继续寻找最小元素，然后放到排序序列末尾。以此类推，直到所有元素均排序完毕。

- 插入排序

> 对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。插入排序在实现上，通常在从后向前扫描过程中，需要反复把已排序元素逐步向后挪位，为最新元素提供插入空间。

- 快速排序

> 步骤:
>
> > 从数列中挑出一个元素，称为 "基准"（pivot），
> > 重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）。在这个分区退出之后，该基准就处于数列的中间位置。这个称为分区（partition）操作。
> > 递归地（recursive）把小于基准值元素的子数列和大于基准值元素的子数列排序。

常见排序代码实现： day3/sort.py

查找

查找(或检索)是在给定信息集上寻找特定信息元素的过程。

二分法查找

当数据量很大适宜采用该方法。采用二分法查找时，数据需是排好序的。

二分查找代码实现： day3/sort.py