计算机考研复试面试常问问题 数据结构篇（上）

使用前需知（拒绝白嫖，如果对你有帮助，你只需点个赞就行）：

> 需要**pdf直接打印版**，可在公众号"**程序员宝藏**"回复**复试上岸**获取(会持续更新)
>
> 
>

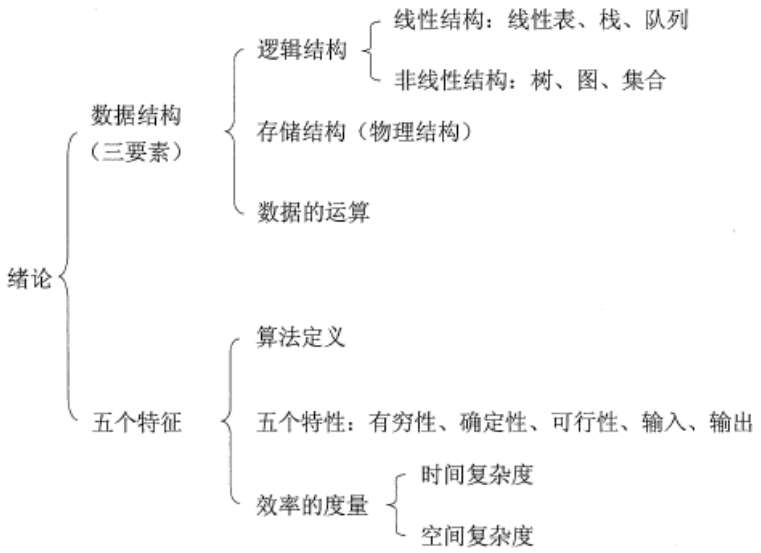
> 在复习过程中，我用心查阅并整理了在**考研复试面试**中可能问到的大部分问题，并**分点整理**了答案，可以**直接理解背诵并加上自己的语言润色**!极力**推荐打印下来看**，效率更高！

> ***声明：一些边边角角的没有收集，毕竟是考研面试，不是笔试，这样也能减轻大家的负担！***

> 此系列一共有8篇：编程语言篇|**数据结构篇**|**操作系统篇**|**组成原理篇**|**计算机网络篇**|**数据库篇**|**软件工程篇**|**计算机专业英语篇(还未全部完成,敬请期待,你们的支持和关注是我最大的动力!**)
>
> 个人整理，不可用于商业用途，转载请注明出处。
>
> 需要**408电子书2021版**，可在公众号"**程序员宝藏**"回复**408电子书**获取
>
> 需要**408初试视频2021版**，可在公众号"**程序员宝藏**"回复**408视频**获取
>
> 需要**复试机试视频**，可在公众号"**程序员宝藏**"回复**机试必过**获取
>
> **加油，大家都可以上岸！！！让我们一起努力！！！**

第一章、绪论

快速唤起记忆知识框架：



---

1 .时间复杂度

> ps:我看到一个学长回忆说老师问了他 大O 是什么意思???如果是我，估计当场蒙蔽了，毕竟平时知道做题就行，也没仔细看 大O 什么意思，所以大家还是看看这题。有备无患。

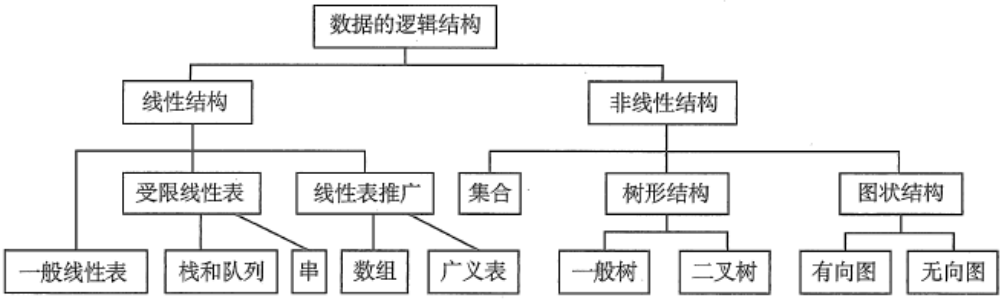
> 一个语句的频度是指该语句在算法中被重复执行的次数。算法中所有语句的频度之和记为T(n), 它是该算法问题规模n 的函数，时间复杂度主要分析T(n) 的数量级。算法中基本运算（最深层循环内的语句）的频度与T(n) 同数量级，因此通常采用算法中基本运算的频度f(n)来分析算法的时间复杂度。因此，算法的时间复杂度记为 T(n) = O(f(n))
>
> > 取f(n) 中随n 增长最快的项，将其系数置为1 作为时间复杂度的度量。例如， $f(n) = an^3 + bn^2 + cn$ 的时向复杂度为$O(n^3)$
>
> 上式中， O 的含义是T(n) 的数量级，其严格的数学定义是：若T(n)和f(n)是定义在正整数集合上的两个函数，则存在正常数C 和n_0，使得当n >= n_0时，都满足0 <=T(n) <=Cf(n) 。
> 算法的时间复杂度不仅依赖于问题的规模n, 也取决于待输入数据的性质（如输入数据元素的初始状态）

2.空间复杂度

> 算法的空间复杂度S(n)定义为该算法所耗费的存储空间，它是问题规模n 的函数。记为
> S(n) = O(g(n))
> 一个程序在执行时除需要存储空间来存放本身所用的指令、常数、变量和输入数据外，还需要一些对数据进行操作的工作单元和存储一些为实现计算所需信息的辅助空间。若输入数据所占空间只取决于问题本身，和算法无关，则只需分析除输入和程序之外的额外空间。
> 算法原地工作是指算法所需的辅助空间为常量，即O(1) 。

3.数的逻辑结构

> 指的是数据元素之间逻辑关系，与数的存储结构无关，是独立于计算机的，以下是分类图。



4.数的存储结构

> 存储结构是指数据结构在计算机中的表示，也称物理结构，主要有以下4种：

> 1) **顺序存储**。把逻辑上相邻的元素存储在物理位置上也相邻的存储单元中，元素之间的关系由存储单元的邻接关系来体现。其**优点**是可以实现随机存取，每个元素占用最少的存储空间；**缺点**是只能使用相邻的一整块存储单元，因此可能产生较多的外部碎片。
>
> 2) **链式存储**。不要求逻辑上相邻的元素在物理位置上也相邻，借助指示元素存储地址的指针来表示元素之间的逻辑关系。其**优点**是不会出现碎片现象，能充分利用所有存储单元；**缺点**是每个元素因存储指针而占用额外的存储空间，且只能实现顺序存取。
>
> 3) **索引存储**。在存储元素信息的同时，还建立附加的索引表。索引表中的每项称为索引项，索引项的一般形式是（关键字，地址）。其**优点**是检索速度快；**缺点**是附加的索引表额外占用存储空间。另外，增加和删除数据时也要修改索引表，因而会花费较多的时间。
>
> 4) **散列存储**。根据元素的关键字直接计算出该元素的存储地址，又称哈希(Hash) 存储。其**优点**是检索、增加和删除结点的操作都很快；**缺点**是若散列函数不好，则可能出现元素存储单元的冲突，而解决冲突会增加时间和空间开销。

5.用循环比递归的效率高吗?

> 循环和递归两者是可以互换的，不能决定性的说循环的效率比递归高。
>
> 递归的优点是：代码简洁清晰，容易检查正确性；缺点是：当递归调用的次数较多时，要增加额外的堆栈处理，有可能产生堆栈溢出的情况，对执行效率有一定的影响。
>
> 循环的优点是：结构简单，速度快；缺点是：它并不能解决全部问题，有的问题适合于用递归来解决不适合用循环。

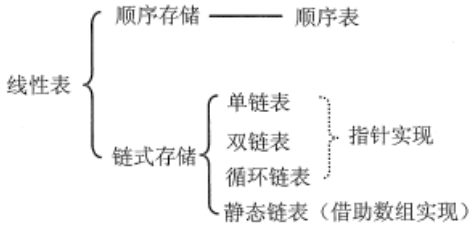
6.贪心算法和动态规划以及分治法的区别?

> **贪心算法**顾名思义就是做出在当前看来是最好的结果，它不从整体上加以考虑，也就是局部最优解。贪心算法从上往下，从顶部一步一步最优，得到最后的结果，它不能保证全局最优解，与贪心策略的选择有关。
>
> **动态规划**是把问题分解成子问题，这些子问题可能有重复，可以记录下前面子问题的结果防止重复计算。动态规划解决子问题，前一个子问题的解对后一个子问题产生一定的影响。在求解子问题的过程中保留哪些有可能得到最优的局部解，丢弃其他局部解，直到解决最后一个问题时也就是初始问题的解。动态规划是从下到上，一步一步找到全局最优解。（各子问题重叠）
>
> **分治法（divide-and-conquer）**：将原问题划分成n个规模较小而结构与原问题相似的子问题；**递归**地解决这些子问题，然后**再合并**其结果，就得到原问题的解。（各子问题独立）
>
> 分治模式在每一层递归上都有三个步骤：

> 分解（Divide）：将原问题分解成一系列子问题；
> 解决（conquer）：递归地解各个子问题。若子问题足够小，则直接求解；
> 合并（Combine）：将子问题的结果合并成原问题的解。
>
> 例如归并排序。

第二章、线性表

快速唤起记忆知识框架：



7.顺序表和链表的比较

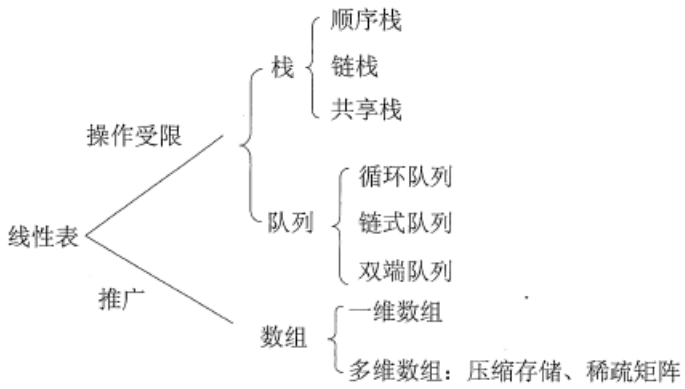
> 1.**存取（读写）方式**
> 顺序表可以顺序存取，也可以随机存取，链表只能从表头顺序存取元素。例如在第i个位置上执行存或取的操作，顺序表仅需一次访问，而链表则需从表头开始依次访问i次。
>
> 2.**逻辑结构与物理结构**
> 采用顺序存储时，逻辑上相邻的元素，对应的物理存储位置也相邻。而采用链式存储时，逻辑上相邻的元素，物理存储位置则不一定相邻，对应的逻辑关系是通过指针链接来表示的。
>
> 3.**查找、插入和删除操作**
> 对于按值查找，顺序表无序时，两者的时间复杂度均为O(n); 顺序表有序时，可采用折半查找，此时的时间复杂度为O(log2n) 。
>
> 对于按序号查找，顺序表支持随机访问，时间复杂度仅为0(1), 而链表的平均时间复杂度为O(n) 。顺序表的插入、删除操作，平均需要移动半个表长的元素。链表的插入、删除操作，只需修改相关结点的指针域即可。由于链表的每个结点都带有指针域，故而存储密度不够大。
>
> 4.**空间分配**
> 顺序存储在静态存储分配情形下，一旦存储空间装满就不能扩充，若再加入新元素，则会出现内存溢出，因此需要预先分配足够大的存储空间。预先分配过大，可能会导致顺序表后部大量闲置；预先分配过小，又会造成溢出。动态存储分配虽然存储空间可以扩充，但需要移动大量元素，导致操作效率降低，而且若内存中没有更大块的连续存储空间，则会导致分配失败。链式存储的结点空间只在需要时申请分配，只要内存有空间就可以分配，操作灵活、高效。

8.头指针和头结点的区别?

> **头指针**：是指向第一个节点存储位置的指针，具有标识作用，头指针是链表的必要元素，无论链表是否为空，头指针都存在。
>
> **头结点**：是放在第一个元素节点之前，便于在第一个元素节点之前进行插入和删除的操作，头结点不是链表的必须元素，可有可无，头结点的数据域也可以不存储任何信息。

第三章、栈和队列

快速唤起记忆知识框架：

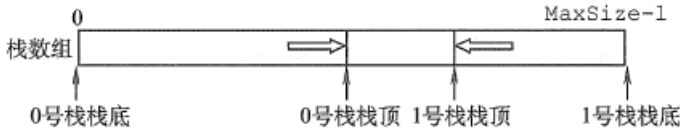


9.栈和队列的区别?

> 队列是允许在一段进行插入另一端进行删除的线性表。队列顾名思义就像排队一样，对于进入队列的元素按“先进先出”的规则处理，在表头进行删除在表尾进行插入。由于队列要进行频繁的插入和删除，一般为了高效，选择用定长数组来存储队列元素，在对队列进行操作之前要判断队列是否为空或是否已满。如果想要动态长度也可以用链表来存储队列，这时要记住队头和对位指针的地址。
>
> 栈是只能在表尾进行插入和删除操作的线性表。对于插入到栈的元素按“后进先出”的规则处理，插入和删除操作都在栈顶进行，与队列类似一般用定长数组存储栈元素。由于进栈和出栈都是在栈顶进行，因此要有一个size变量来记录当前栈的大小，当进栈时size不能超过数组长度，size+1，出栈时栈不为空，size-1。

10.共享栈

> 利用栈底位置相对不变的特性，可以让两个顺序栈共享一个一维数组空间，将两个栈的栈底分别设置在共享空间的两端，两个栈顶向共享空间的中间延伸。这样能够更有效的利用存储空间，两个栈的空间相互调节，只有在整个存储空间被占满时才发生上溢。



11.如何区分循环队列是队空还是队满?

> 普通情况下，循环队列队空和队满的判定条件是一样的，都是Q.front == Q.rear。
>
> ps:队头指针指向第一个数；队尾指针指向最后一个数的下一个位置，即将要入队的位置。

> 方法一：牺牲一个单元来区分队空和队满，这个时候(Q.rear+1)%MaxSize == Q.front才是队满标志 。

> 方法二：类型中增设表示元素个数的数据成员。这样，队空的条件为Q.size == 0;队满的条件为Q.size == MaxSize。

12.栈在括号匹配中的算法思想?

> 括号匹配算法思想

（1）出现的凡是“左括号”，则进栈；

（2）出现的是“右括号”，

首先检查栈是否空？

若栈空，则表明该“右括号”多余

否则和栈顶元素比较？

若相匹配，则栈顶“左括号出栈”

否则表明不匹配

（3）表达式检验结束时，

若栈空，则表明表达式中匹配正确

否则表明“左括号”有余；

13.栈在通过后缀表达式求值的算法思想？

顺序扫描表达式的每一项，然后根据它的类型做如下相应操作：若该项是操作数，则将其压入栈中；若该项是操作符,则连续从栈中退出两个操作数y 和x, 形成运算指令XY, 并将计算结果重新压入栈中。当表达式的所有项都扫描并处理完后，栈顶存放的就是最后的计算结果。

表 3.1　后缀表达式 ABCD-*+EF/-求值的过程

步	扫描项	项类型	动　作	栈中内容
1			置空栈	空
2	A	操作数	进栈	A
3	B	操作数	进栈	A B
4	C	操作数	进栈	A B C
5	D	操作数	进栈	A B C D
6	-	操作符	D、C 退栈，计算 C-D，结果 R_1 进栈	A B R_1
7	*	操作符	R_1、B 退栈，计算 B×R_1，结果 R_2 进栈	A R_2
8	+	操作符	R_2、A 退栈，计算 A+R_2，结果 R_3 进栈	R_3
9	E	操作数	进栈	R_3 E
10	F	操作数	进栈	R_3 E F
11	/	操作符	F、E 退栈，计算 E/F，结果 R_4 进栈	R_3 R_4
12	-	操作符	R_4、R_3 退栈，计算 R_3-R_4，结果 R_5 进栈	R_5

14.栈在递归中的应用？

递归是一种重要的程序设计方法。简单地说，若在一个函数、过程或数据结构的定义中又应用了它自身，则这个函数、过程或数据结构称为是递归定义的，简称递归。
它通常把一个大型的复杂问题层层转化为一个与原问题相似的规模较小的问题来求解，递归策略只需少量的代码就可以描述出解题过程所需要的多次重复计算，大大减少了程序的代码量。但在通常情况下，它的效率并不是太高。

将递归算法转换为非递归算法，通常需要借助栈来实现这种转换。

15.队列在层次遍历中的作用？

在信息处理中有一大类问题需要逐层或逐行处理。这类问题的解决方法往往是在处理当前层或当前行时就对下一层或下一行做预处理，把处理顺序安排好，待当前层或当前行处理完毕，就可以处理下一层或下一行。使用队列是为了保存下一步的处理顺序。下面用二叉树层次遍历的例子，说明队列的应用。

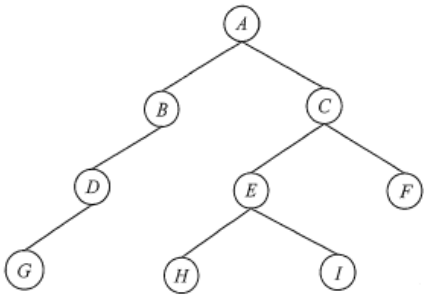


图 3.17　二叉树

序	说　明	队内	队　外
1	*A* 入	*A*	
2	*A* 出，*BC* 入	*BC*	*A*
3	*B* 出，*D* 入	*CD*	*AB*
4	*C* 出，*EF* 入	*DEF*	*ABC*
5	*D* 出，*G* 入	*EFG*	*ABCD*
6	*E* 出，*HI* 入	*FGHI*	*ABCDE*
7	*F* 出	*GHI*	*ABCDEF*
8	*GHI* 出		*ABCDEFGHI*

16.队列在计算机系统中的应用?

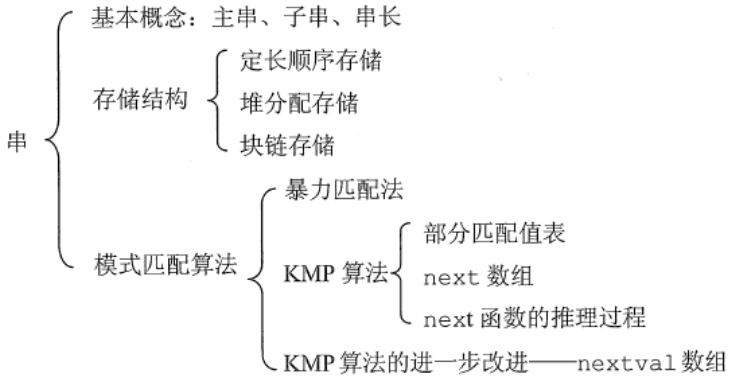
> 队列在计算机系统中的应用非常广泛，以下仅从两个方面来简述队列在计算机系统中的作用：第一个方面是解决主机与外部设备之间速度不匹配的问题，第二个方面是解决由多用户引起的资源竞争问题。
>
> 对于第一个方面，仅以主机和打印机之间速度不匹配的问题为例做简要说明。主机输出数据给打印机打印，输出数据的速度比打印数据的速度要快得多，由于速度不匹配，若直接把输出的数据送给打印机打印显然是不行的。解决的方法是设置一个打印数据缓冲区，主机把要打印输出的数据依次写入这个缓冲区，写满后就暂停输出，转去做其他的事情。打印机就从缓冲区中按照先进先出的原则依次取出数据并打印，打印完后再向主机发出请求。主机接到请求后再向缓冲区写入打印数据。这样做既保证了打印数据的正确，又使主机提高了效率。由此可见，打印数据缓冲区中所存储的数据就是一个队列。
>
> 对于第二个方面， CPU (即中央处理器，它包括运算器和控制器）资源的竞争就是一个典型的例子。在一个带有多终端的计算机系统上，有多个用户需要CPU 各自运行自己的程序，它们分别通过各自的终端向操作系统提出占用CPU 的请求。操作系统通常按照每个请求在时间上的先后顺序，把它们排成一个队列，每次把CPU 分配给队首请求的用户使用。当相应的程序运行结束或用完规定的时间间隔后，令其出队，再把CPU 分配给新的队首请求的用户使用。这样既能满足每个用户的请求，又使CPU 能够正常运行。

17.矩阵的压缩存储

> 数据结构中，提供针对某些特殊矩阵的压缩存储结构。这里所说的特殊矩阵，主要分为以下两类：
>
> - 含有大量相同数据元素的矩阵，比如对称矩阵；
> - 含有大量 0 元素的矩阵，比如稀疏矩阵、上（下）三角矩阵；
>
> 针对以上两类矩阵，数据结构的压缩存储思想是：矩阵中的相同数据元素（包括元素 0）只存储一个。

第四章、串

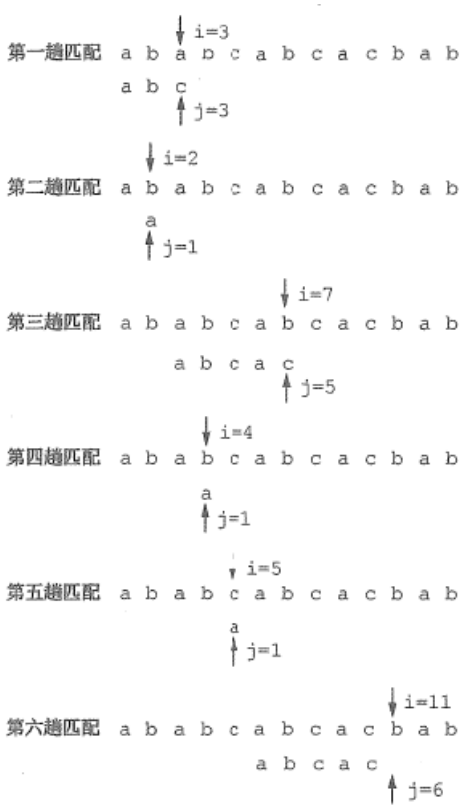
快速唤起记忆知识框架：



18.串的模式匹配

> 子串的定位操作通常称为串的模式匹配，他求的是子串（常称模式串）在主串中的位置。

> **暴力模式匹配算法**的思想是：从主串的第一个字符起，与子串的第一个字符比较，相等则继续比较；不等则从主串的下一个位置起，继续和子串开始比较，直到最后看是否匹配成功。
>
> 以下的子串为：‘abcac’:



> **改进的模式匹配算法-----KMP算法：**
>
> 在暴力匹配中，每趟匹配失败都是模式后移一位再从头开始比较。而某趟已匹配相等的字符序列是模式的某个前缀，这种频繁的重复比较相当于模式串在不断地进行自我比较，这就是其低效率的根源。因此，可以从分析模式本身的结构着手，如果已匹配相等的前缀序列中有某个后缀正好是模式的前缀，那么就可以将模式向后滑动到与这些相等字符对齐的位置，主串i指针无须回溯，并继续从该位置开始进行比较。而模式向后滑动位数的计算仅与模式本身的结构有关，与主串无关。
>
> **（此处篇幅过长，感兴趣可以直接百度KPM算法详细了解，当然要求不高的可以直接跳过，知道大致思想即可，不要被问到一问三不知就可以）**