软件设计师考试背记精要

1、 数组与矩阵：

数组类型	存储地址计算
一维数组a[n]	a[i]的存储地址为：a+i*len
二维数组a[m][n]	a[i][j]的存储地址（按行存储）为：a+(i*n+j)*len a[i][j]的存储地址（按列存储）为：a+(j*m+i)*len

2、顺序表与链表对比：

性能类别	具体项目	顺序存储	链式存储
空间性能	存储密度	=1，更优	<1
	容量分配	事先确定	动态改变，更优
时间性能	查找运算	O(n/2)	O(n/2)
	读运算	O(1)，更优	O([n+1]/2)，最好情况为1，最坏情况为n
	插入运算	O(n/2)，最好情况为0，最坏情况为n	O(1)，更优
	删除运算	O([n-1]/2)	O(1)，更优

3、循环链表：队空条件：head=tail；队满条件：(tail+1)%size=head。

4、树的概念：

（1） 双亲、孩子和兄弟：结点的子树的根称为该结点的孩子；相应地，该结点称为其子结点的双亲。具有相同双亲的结点互为兄弟。

（这里涉及到 2 个层次，第一个层次的子树，这棵子树的根是第一层结点的孩子结点，第一层结点是其子节点的双亲节点/父节点）。

（2） 结点的度：一个结点的子树的个数记为该结点的度。

（3） 叶子节点：也称为终端结点，指度为 0 的结点。

（4） 内部结点：指度不为 0 的结点，也称为分支节点或非终端节点。除根结点之外，分支结点也称为内部结点。

（5） 结点的层次：根为第一层，根的孩子为第二层，依次类推，若某节点在第 i 层，则其孩子结点在第 i+1 层。

（6） 树的高度：一颗树的最大层次数记为树的高度（深度）。

5、二叉树的重要特性：

（1） 在二叉树的第 i 层上最多有2^{i-1}个结点（i≥1）。

（2） 深度为 k 的二叉树最多有2^k -1 个结点（k≥1）。

（3） 对任何一棵二叉树，如果其叶子结点数为 n0，度为 2 的结点数为 n2，则 n0=n2+1。

（4） 如果对一棵有 n 个结点的完全二叉树的结点按层序编号（从第 1 层到 L$\log_2 n$ 」+1 层，每层从左到右），则对任一结点 i（1≤i≤n），有：

如果 i=1，则结点 i 无父结点，是二叉树的根；如果 i>1，则父结点是 L i/2 」；

如果 2i>n，则结点 i 为叶子结点，无左子结点；否则，其左子结点是结点 2i；

如果 2i+1>n，则结点 i 无右子叶点，否则，其右子结点是结点 2i+1。

6、特殊的树：

（1） 二叉树：二叉树是每个结点最多有两个孩子的有序数，可以为空树，可以只有一个结点。

（2） 满二叉树：任何结点，或者是树叶，或者恰有两棵非空子树。

（3） 完全二叉树：最多只有最下面的两层结点的度可以小于 2，并且最下面一层的结点全都集中在该层左侧的若干位置。

（4） 平衡二叉树：树中任一结点的左右子树高度之差不超过 1。

（5） 查找二叉树：又称之为排序二叉树。任一结点的权值，大于其左孩子结点，小于其右孩子结点。

（6） 线索二叉树：在每个结点中增加两个指针域来存放遍历时得到的前驱和后继信息。

（7） 最优二叉树：又称为哈弗曼树，它是一类带权路径长度最短的树。

7、最优二叉树（哈弗曼树）的构造过程：（1）根据给定的权值集合，找出最小的两个权值，构造一棵子树将这两个权值作为其孩子结点，二者权值之和作为根结点；（2）在原集合中删除这两个结点的权值，并引入根节点的权值；（3）重复步骤（1）和步骤（2），直到原权值集合为空。

8、二叉树的遍历：遍历是按某种策略访问树中的每个结点，且仅访问一次的过程。

（1） 前序遍历：又称为先序遍历，按根 -> 左 -> 右的顺序进行遍历。

（2） 后序遍历：按左 -> 右 -> 根的顺序进行遍历。

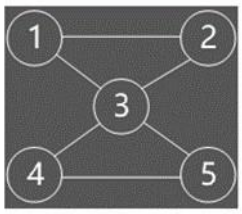
（3） 中序遍历：按左 -> 根 -> 右的顺序进行遍历。

（4） 层次遍历：按层次顺序进行遍历。

9、图的邻接矩阵表示：用一个 n 阶方阵 R 来存放图中各结点的关联信息，其矩阵元素 Rij 定义为：

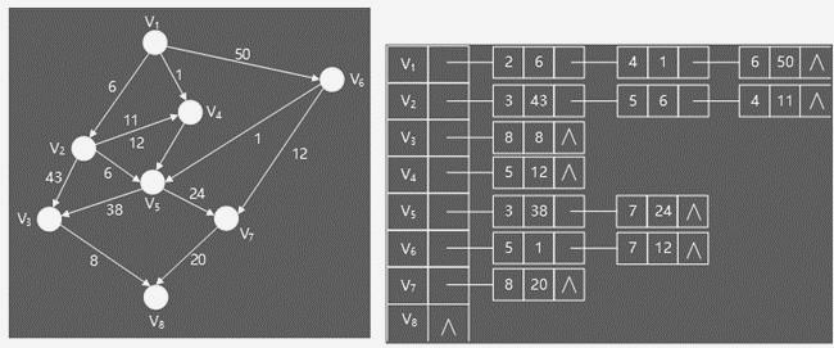
$$R_{ij}=\begin{cases}1 & \text{若顶点i到顶点j有邻接边}\\ 0 & \text{若顶点i到顶点j无邻接边}\end{cases}$$

如：

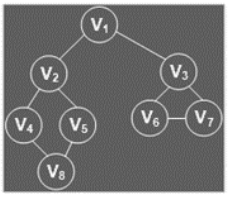


$$R_1=\begin{bmatrix}0&1&1&0&0\\1&0&1&0&0\\1&1&0&1&1\\0&0&1&0&1\\0&0&1&1&0\end{bmatrix}$$

10、图的邻接表表示：首先把每个顶点的邻接顶点用链表示出来，然后用一个一维数组来顺序存储上面每个链表的头指针。如：



11、图的遍历：

遍历方法	说明	示例	图例
深度优先	1.首先访问出发顶点V 2.依次从V出发搜索V的任意一个邻接点W； 3.若W未访问过，则从该点出发继续深度优先遍历 它类似于树的前序遍历。	V_1，V_2，V_4，V_8，V_5，V_3，V_6，V_7	
广度优先	1.首先访问出发顶点V 2.然后访问与顶点V邻接的全部未访问顶点W、X、Y…； 3.然后再依次访问W、X、Y…邻接的未访问的顶点；	V_1，V_2，V_3，V_4，V_5，V_6，V_7，V_8	

12、图的拓扑排序：拓扑排序是将 AOV 网中的所有顶点排成一个线性序列的过程，并且该序列满足：若在 AOV 网点中从顶点 Vi 到 Vj 有一条路径，则在该线性序列中，顶点 Vi 必然在顶点 Vj 之前。

13、顺序查找的思想：将待查找的关键字为 key 的元素从头到尾与表中元素进行比较，如果中间存在关键字为 key 的元素，则返回成功；否则，则查找失败。

14、A 二分法查找的基本思想是：（设 R[low,…,high]是当前的查找区）

（1） 确定该区间的中点位置：mid=L(low+high)/2」;

（2） 将待查的 k 值与 R[mid].key 比较，若相等，则查找成功并返回此位置，否则需确定新的查找区间，继续二分查找，具体方法如下：

若 R[mid].key > k，则由表的有序性可知 R[mid,…,n].key 均大于 k，因此若表中存在关键字等于 k 的结点，则该结点必定是在位置 mid 左边的子表 R[low,…,mid−1]中。因此，新的查找区间是左子表 R[low,…,high]，其中 high=mid−1。

若 R[mid].key<k，则要查找的 k 必在 mid 的右子表 R[mid+1,…,high]中，即新的查找区间是右子表 R[low,…,high]，其中 low=mid+1。

若 R[mid].key=k，则查找成功，算法结束。

（3） 下一次查找是针对新的查找区间进行，重复步骤（1）和（2）。

（4） 在查找过程中，low 逐步增加，而 high 逐步减少。如果 high<low，则查找失败，算法结束。

折半查找在查找成功时关键字的比较次数最多为 L $\log_2 n$ +1 」 次。

折半查找的时间复杂度为 O($\log_2 n$)次。

16、散列表查找的基本思想是：已知关键字集合 U，最大关键字为 m，设计一个函数 Hash，它以关键字为自变量，关键字的存储地址为因变量，将关键字映射到一个有限的、地址连续的区间 T[0..n-1](n<<m)中，这个区间就称为散列表，散列查找中使用的转换函数称为散列函数。

18、各种排序算法对比：

类 别	排序方法	时间复杂度		空间复杂度	稳 定 性
		平均情况	最坏情况	辅助存储	
插入排序	直接插入	$O(n^2)$	$O(n^2)$	O(1)	稳定
	Shell排序	$O(n^{1.3})$	$O(n^2)$	O(1)	不稳定
选择排序	直接选择	$O(n^2)$	$O(n^2)$	O(1)	不稳定
	堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	O(1)	不稳定
交换排序	冒泡排序	$O(n^2)$	$O(n^2)$	O(1)	稳定
	快速排序	$O(n\log_2 n)$	$O(n^2)$	$O(\log_2 n)$	不稳定
归并排序		$O(n\log_2 n)$	$O(n\log_2 n)$	O(n)	稳定
基数排序		O(d(r+n))	O(d(r+n))	O(r+n)	稳定

19、排序算法应用情景对比：

（1） 若待排序列的记录数目 n 较小，可采用直接插入排序和简单选择排序。由于直接插入排序所需的记录移动操作较简单选择排序多，因而当记录本身信息量大时，用简单选择排序方法较好。

（2） 若待排记录按关键字基本有序，宜采用直接插入排序或冒泡排序。

（3） 当 n 很大且关键字位数较少时，采用基数排序较好。

（4） 若 n 很大，则应采用时间复杂度为 O($n\log_2 n$)的排序方法，例如快速排序、堆排序或归并排序：

快速排序目前被认为是内部排序中最好的方法，当待排序的关键字为随机分布时，快速排序的平均运行时间最短；

堆排序只需要一个辅助空间，并且不会出现在快速排序中可能出现的最快情况。

快速排序和堆排序都是不稳定的排序方法，若要求排序稳定，可选择归并排序。

20、常见的对算法执行所需时间的度量：

$O(1)<O(\log_2 n)<O(n)<O(n\log_2 n)<O(n^2)<O(n^3)<O(2^n)$

21、常见算法逻辑的时间复杂度：

（1） 单个语句，或程序无循环和复杂函数调用：O(1)。

（2） 单层循环：O(n)；双层嵌套循环：$O(n^2)$；三层嵌套循环：$O(n^3)$。

（3） 树形结构、二分法、构建堆过程：$O(\log_2 n)$。

（4） 堆排序、归并排序：$O(n\log_2 n)$。

（5） 所有不同可能的排列组合：$O(2^n)$。

22、分治法：把一个问题拆分成多个小规模的相同子问题，一般可用递归解决。

23、动态规划法：划分子问题（最优子结构），并把子问题结果使用数组存储，利用查询子问题结果构造最终问题结果。

24、贪心法：局部最优，但整体不见得最优。每步有明确的，既定的策略。

25、回溯法：系统的搜索一个问题的所有解或任一解。有试探和回退的过程。

26、面向对象基本概念：

对象：属性（数据）+方法（操作）+对象 ID

封装：隐藏对象的属性和实现细节,仅对外公开接口（信息隐藏技术）

类（实体类/控制类/边界类）

接口：一种特殊的类，他只有方法定义没有实现

继承与泛化：复用机制（单重继承和多重继承）

重置/覆盖（Overriding）：在子类中重新定义父类中已经定义的方法

重载：一个类可以有多个同名而参数类型不同的方法

多态：不同对象收到同样的消息产生不同的结果

过载多态：同一个名字在不同的上下文中所代表的含义不同

动态绑定：根据接收对象的具体情况将请求的操作与实现的方法进行连接（运行时绑定）

消息和消息通信：对象之间进行通信的一种构造叫做消息。消息是异步通信的（消息传递：接收到信息的对象经过解释，然后予以响应）

下午题考试做题技巧：

第一大题数据流图：

考试形式：

（1） 补充外部实体、数据存储或者是加工名。

要注意这些补充的信息都来源于题干的描述，注意查找关键词即可，补充外部实体：人物角色（客户、管理员、主管、经理、老师、学生）、组织机构（银行、供应商、募捐机构）、外部系统（银行系统、工资系统、后台数据库）；补充存储：“...文件”，“...表”，“....库”，“.. 清单”；补充加工名：一般是由“动词+名词”的形式组成。

（2） 补充数据流。

补充的数据流需要应用的技巧包含两个方面：第一个遵循的就是数据流图平衡原则，通过数据流图平衡原则可以判定实体之间是否还存在不平衡的问题，如果保持平衡接下来就需要根据题干描述，这一类一般是数据存储和加工之间的数据流向，要特别注意数据流必须要加工有关，即数据流的至少有一端为加工。

（3） 主观性质答题，可能出现的情况：结构化语言、判断平衡原则出现的问题（黑洞、奇迹、灰洞）、分解加工、判断外部的信息组成，大家只要根据所学习的知识进行答题即可，合理即给分。

第二大题数据库设计

考试形式：

（1）补充联系，三种类型（1：1,1：*，*：*）。

根据题目的描述即可，比如：“每个部门可以有多名员工，每名员工只属于一个部门”，部门和员工之间存在 1：*的联系。补充的联系需要通过画的方式画在答题卡上。

（2） 补充属性。

根据题目的描述，依据的是 ER 联系和归并的过程和对应题干对于该属性的描述，从而进行补充。

依据的 ER 联系的归并过程分为（1：1，1：*和*：*）三种情况，对应的转换规则需要熟悉。

（3） 判断主、外键

主键一般题干描述：“***唯一标识”，提到这类关键字眼就是属于主键，其次根据前面提到的 ER 联系归并过程来确定主键，例如：多对多的归并过程就是多端和多端的结合主键。外键就是其他关系的主键，是属于其他主键归并的过来的情况。

（4） 主观性质答题，可能出现的例题类型：判断数据库是否存在规范化问题（根据前面学习的规范法理论），数据库存在什

么不合理性，如何进行设计，补充什么样的实体，增加什么样的联系等等，这类主观题型，只要根据自己的思路去答，都能够得到相应的分数。

第三大题 UML 建模

考试形式：

（1） 补充对应图示的名称，一般来说是两个 UML 图示进行结合考查，补充的类图的类名（用名词表示），用例图的用例名（动名词），状态图和活动图的状态名或者活动名，这类补充的名称根据题干描述和对应的各个图示的关系来判断，（类图对应的关系泛化关系、聚合关系、组合关系、实现关系、依赖关系）等，（用例图泛化关系、包含关系、扩展关系）等。

（2） 补充类中的一些方法名，或者是增加一些的新的用例描述，已经关于某个类的组成部分这类考查形式。这类题需要根据题目描述通过学习的技巧去关联，比方说对于方法名的填写，需要根据的具体箭头指向，箭头指向谁，表示这个方法属于这个类里面的。

（3） 会结合设计模式或者是一些其他类型的主观性质的题型，根据题目要求去补充一些对应的东西，需要根据平时积累的知识和依据对于 UML 的理解去作答，合理即可得到相应的分数。

第四大题数据结构与算法基础

考试形式：

（1） 代码填空方式，一般来说是四个代码填空，共计 8 分，这部分的考查语法主要来自于对 C 语言的基础知识进行考查，需要掌握一定的填空技巧，有 2 空左右是可以轻松填出来的，一般来自题干给出的信息和对应的值做替换和修正等操作。

剩下的 2 空需要大家具有一定的基础和对代码的理解，难度较大。

（2） 算法策略和时间复杂度的填写。四种算法策略的判断：

分治法：把一个问题拆分成多个小规模的相同子问题，一般可以用递归解决。

动态规划法：强调“最优子结构”和递归式，一般自顶向下的时间复杂度为 O（2^n），自底向上为 O（n^a）。

回溯法：系统搜索一个问题的所有解或任一解，一般有回退的效果，我们的深度优先就属于回溯法。

贪心算法：是属于典型局部最优，但不见得是整体最优，每步都有明确的，既定的策略。

时间复杂度的判断一般根据特殊的 for 循环的层数判断，对于比较熟悉的分治法就是 O（log2n）。

（3） 给出代码的运行结果。这部分难度较大，一般来说需要对于第一问进行详细解读，如果没有这个基础可以做适当性的放弃。

第五大题面向对象程序设计（JAVA 或 C++）

考试形式：

（1）5 个填空，每空 3 分，共计 15 分，要求大家的分值在 6-9 分即可，对于要学习的内容包含面向对象的基本概念，常见的修饰符（+表示 public，#表示 protected，-表示 private），以及类名，接口，抽象方法等基本概念，熟悉对象实例化，方法调用，设计模式的形式（23 种设计模式）等。