| | | | | | |
|---|---|---|---|---|---|
| “数组”的应用 | 对称矩阵的压缩存储 | 1.1.1 | 给自己出题：自己动手创造，画一个5行5列的对称矩阵 | | 尚未在应用题中考过“对称矩阵压缩存储”<br>未来应用题有可能将无向图的邻接矩阵、对称矩阵压缩存储一起考察 |
| | | 1.1.2 | 画图：按“行优先”压缩存储上述矩阵，画出一维数组的样子 | | |
| | | 1.1.3 | 简答：写出元素 $i,j$ 与 数组下标之间的对应关系 | | |
| | | 1.1.4 | 画图：按“列优先”压缩存储上述矩阵，画出一维数组的样子 | | |
| | | 1.1.5 | 简答：写出元素 $i,j$ 与 数组下标之间的对应关系 | | |
| | | 1.1.6 | 画图：假设你的对称矩阵表示一个无向图，画出无向图的样子 | | |
| | 上/下三角矩阵的压缩存储 | 1.2.1 | 给自己出题：自己动手创造，画一个5行5列的下三角矩阵 | | 2011 年 41 题曾考过“三角矩阵的压缩存储” |
| | | 1.2.2 | 画图：按“行优先”压缩存储上述矩阵，画出一维数组的样子 | | |
| | | 1.2.3 | 简答：写出元素 $i,j$ 与 数组下标之间的对应关系 | | |
| | | 1.2.4 | 画图：按“列优先”压缩存储上述矩阵，画出一维数组的样子 | | |
| | | 1.2.5 | 简答：写出元素 $i,j$ 与 数组下标之间的对应关系 | | |
| | | 1.2.6 | 画图：假设你的对称矩阵表示一个有向图，画出有向图的样子 | | |
| | 三对角矩阵的压缩存储 | 1.3.1 | 给自己出题：自己动手创造，画一个5行5列的三对角矩阵 | | “三对角矩阵”在应用题中的可考察性较弱，很难和其他考点一起综合考，更可能考小题 |
| | | 1.3.2 | 画图：按“行优先”压缩存储上述矩阵，画出一维数组的样子 | | |
| | | 1.3.3 | 简答：写出元素 $i,j$ 与 数组下标之间的对应关系 | | |
| | | 1.3.4 | 画图：按“列优先”压缩存储上述矩阵，画出一维数组的样子 | | |
| | | 1.3.5 | 简答：写出元素 $i,j$ 与 数组下标之间的对应关系 | | |

| | | | | | |
|---|---|---|---|---|---|
| “栈、队列”的应用 | 栈的定义和基本操作实现 | 2.1.1 | 写代码：定义顺序存储的栈（数组实现），数据元素是 int 型 | | 截至2023年，历年真题皆未考过“栈的应用”，因此该部分很可能是未来的应用题出题点。 |
| | | 2.1.2 | 写代码：基于上述定义，实现“出栈、入栈、判空、判满”四个基本操作 | | |
| | | 2.1.3 | 写代码：定义链式存储的栈（单链表实现） | | |
| | | 2.1.4 | 写代码：基于上述定义，栈顶在链头，实现“出栈、入栈、判空、判满”四个基本操作 | | |
| | | 2.1.5 | 写代码：定义链式存储的栈（双向链表实现） | | |
| | | 2.1.6 | 写代码：基于上述定义，栈顶在链尾，实现“出栈、入栈、判空、判满”四个基本操作 | | |
| | | 2.1.7 | 给自己出题：自己动手创造，写一个具有多层小括号、中括号的算数表达式 | | |
| | | 2.1.8 | 画图：针对2.1.7的算数表达式，使用栈进行“括号匹配”，画出栈内元素最多的状态 | | |
| | | 2.1.9 | 简答：请描述使用栈进行括号匹配的过程 | | |
| | 队列的定义和基本操作实现 | 2.2.1 | 写代码：定义顺序存储的队列（数组实现），要求数组空间可以被循环利用 | | 2019年42题考过“队列的应用”，个人认为近几年应用题再次考“队列”的可能性低于“栈” |
| | | 2.2.2 | 写代码：基于上述定义，实现“出队、入队、判空、判满”四个基本操作 | | |
| | | 2.2.3 | 写代码：定义链式存储的队列（单链表实现） | | |
| | | 2.2.4 | 写代码：基于上述定义，实现“出队、入队、判空、判满”四个基本操作 | | |

| | | | | | |
|---|---|---|---|---|---|
| 树的应用 | 二叉树的性质 | 3.1.1 | 总结二叉树的度、树高、结点数等属性之间的关系 | | 通过王道书 5.2.3 课后小题来复习“二叉树的性质” |
| | 二叉树的顺序存储和基本操作实现 | 3.1.2 | 写代码：定义顺序存储的二叉树（数组实现，树的结点从数组下标1开始存储） | | 这些“基本操作”，可以将顺序存储的二叉树映射为链式存储的二叉树。有助于大家解决“顺序存储二叉树”的算法题 |
| | | 3.1.3 | 基于上述定义，写一个函数 int findFather ( i )，返回结点 i 的父节点编号 | | |
| | | 3.1.4 | 基于上述定义，写一个函数 int leftChild ( i )，返回结点 i 的左孩子编号 | | |
| | | 3.1.5 | 基于上述定义，写一个函数 int rightChild ( i )，返回结点 i 的右孩子编号 | | |
| | | 3.1.6 | 利用上述三个函数，实现先/中/后序遍历 | | |
| | | 3.1.7 | 写代码：定义顺序存储的二叉树（数组实现，树的结点从数组下标0开始存储） | | |
| | | 3.1.8 | 基于上述定义，写一个函数 int findFather ( i )，返回结点 i 的父节点编号 | | |
| | | 3.1.9 | 基于上述定义，写一个函数 int leftChild ( i )，返回结点 i 的左孩子编号 | | |
| | | 3.1.10 | 基于上述定义，写一个函数 int rightChild ( i )，返回结点 i 的右孩子编号 | | |
| | | 3.1.11 | 利用上述三个函数，实现先/中/后序遍历 | | |
| | 树的性质 | 3.2.1 | 总结树的度、树高、结点数等属性之间的关系 | | 通过王道书 5.1.4、5.4.4 课后小题来复习“树和森林的性质” |
| | 树（森林）的定义和画图 | 3.3.1 | 写代码：使用“双亲表示法”，定义顺序存储的树（以及森林） | | 至今为止还没在“应用题”考过树和森林，但这部分非常符合408应用题的风格。要特别注意。 |
| | | 3.3.2 | 写代码：使用“孩子表示法”，定义链式存储的树（以及森林） | | |
| | | 3.3.3 | 对比：树的孩子表示法存储 v.s. 图的邻接表存储 v.s. 散列表的拉链法 v.s. 基数排序。你发现了什么? | | |
| | | 3.3.4 | 写代码：使用“孩子兄弟表示法”，定义链式存储的树（以及森林） | | |
| | | 3.3.5 | 自己动手创造，画一个结点总数不少于10的树，并画出对应的“双亲表示法、孩子表示法、孩子兄弟表示法”三种数据结构的示意图 | | |
| | | 3.3.6 | 自己动手创造，画一个至少包含3棵树的森林，并画出对应的“双亲表示法、孩子表示法、孩子兄弟表示法”三种数据结构的示意图 | | |
| | 哈夫曼树的应用 | 3.4.1 | 自己动手创造，写10个字符，并给每个字符设置权值，画出构造哈夫曼树的过程 | | 哈夫曼树在历年真题中考察次数较多，可以通过历年真题来复习“哈夫曼树”相关知识。今年的大题再次考哈夫曼树的概率较低 |
| | | 3.4.2 | 用文字描述构造哈夫曼树的过程 | | |
| | | 3.4.3 | 基于你所构造的哈夫曼树，写出10个字符的哈夫曼编码 | | |
| | | 3.4.4 | 用文字描述根据一棵哈夫曼树“译码”的过程（即如何将二进制哈夫曼编码翻译为字符） | | |
| | 并查集的应用 | 3.5.1 | 写代码：定义一个并查集（用长度为n的数组实现） | | 并查集是2022年大纲新增考点，有史以来还没有考过。今年要特别注意，很有可能在小题、大题中考察（考应用题的可能性较大） |
| | | 3.5.2 | 基于上述定义，实现并查集的基本操作——并 Union | | |
| | | 3.5.3 | 基于上述定义，实现并查集的基本操作——查 Find | | |
| | | 3.5.4 | 自己设计一个例子，并查集初始有10个元素，进行若干次Union操作，画出每一次Union后的样子 | | |
| | | 3.5.5 | 自己设计一个例子，基于上一步得到的并查集，进行若干次find操作（每次find会进行“路径压缩”）。画出每次 find （路径压缩）后的样子 | | |
| | 二叉排序树、平衡二叉树的应用题潜在考法 | 3.6.1 | 自己设计一个例子，给出不少于10个关键字序列，按顺序插入一棵初始为空的二叉排序树，画出每一次插入后的样子 | | 截至目前，尚未考过二叉排序树/平衡二叉树的插入、删除、查找，也没有考过 ASL 的计算。这个部分很有可能是今年“应用题”的出题点，一定要做好训练。 |
| | | 3.6.2 | 基于你设计的例子，计算二叉排序树在查找成功和查找失败时的 ASL | | |
| | | 3.6.3 | 基于你设计的例子，依次删除不少于4个元素，画出每一次删除之后的样子（需要包含四种删除情况——删一个叶子结点、删一个只有左子树的结点、删一个只有右子树的结点、删一个既有左子树又有右子树的结点） | | |
| | | 3.6.4 | 自己设计一个例子，给出不少于10个关键字序列，按顺序插入一棵初始为空的平衡二叉树，画出每一次插入后的样子（你设计的例子要涵盖LL、RR、LR、RL四种调整平衡的情况） | | |
| | | 3.6.5 | 基于你设计的例子，计算平衡二叉树在查找成功和查找失败时的 ASL | | |

| | | | | | |
|---|---|---|---|---|---|
| 图的应用 | 图的性质 | 4.1.1 | 总结无向图、有向图的结点数、边数、度数、连通性、强连通性等性质之间的关系 | | 通过王道书 6.1.2 课后小题来复习“图的性质” |
| | 图的数据结构定义 | 4.2.1 | 写代码：定义一个顺序存储的图（邻接矩阵实现） | | 大纲要求掌握图的四种存储方法，分别是“邻接矩阵、邻接表、邻接多重表、十字链表”。其中，邻接矩阵和邻接表更有可能在应用题中进行考察，需要掌握数据结构定义和画图 |
| | | 4.2.2 | 写代码：定义一个链式存储的图（邻接表实现） | | |
| | | 4.2.3 | 自己设计一个不少于6个结点的带权无向图，并画出其邻接矩阵、邻接表的样子 | | |
| | | 4.2.4 | 自己设计一个不少于6个结点的带权有向图，并画出其邻接矩阵、邻接表的样子 | | |
| | 图的应用：最小生成树 | 4.3.1 | 自己设计一个不少于6个结点的带权无向连通图，并画出其邻接矩阵、邻接表的样子 | | 最小生成树在以前的真题中已经考过两次应用题（2017、2018），而且考察难度不大。 |
| | | 4.3.2 | 基于上述无向连通图，使用Prim算法生成MST，画出算法执行过程的示意图，并计算MST的总代价 | | |
| | | 4.3.3 | 基于上述无向连通图，使用Kruskal算法生成MST，画出算法执行过程的示意图，并计算MST的总代价 | | |
| | 图的应用：最短路径 | 4.4.1 | 基于你设计的带权有向图，从某一结点出发，执行Dijkstra算法求单源最短路径。用文字描述每一轮执行的过程 | | 最短路径问题在以前的真题中已经考过两次应用题（2014、2009） |
| | | 4.4.2 | 文字描述：用BFS算法求单源最短路径的过程 | | |
| | 图的应用：拓扑排序 | 4.5.1 | 自己设计一个不少于6个结点的带权有向无环图，并画出其邻接矩阵的样子 | | 截至目前，拓扑排序尚未在“应用题”中考过，需要特别注意。<br>如果一个有向图可以进行拓扑排序，就一定可以用上三角/下三角邻接矩阵存储。因此“拓扑排序”很可能和矩阵的压缩存储结合起来考察。 |
| | | 4.5.2 | 用一维数组将你设计的有向无环图的邻接矩阵进行压缩存储 | | |
| | | 4.5.3 | 文字描述：基于你压缩存储的数组，如何判断结点 i、j 之间是否有边? | | |
| | | 4.5.4 | 基于你设计的带权有向无环图，写出所有合法的拓扑排序序列 | | |
| | | 4.5.5 | 文字描述：拓扑排序的过程 | | |
| | 图的应用：关键路径 | 4.6.1 | 基于你设计的带权有向无环图，写出所有合法的关键路径，并算出关键路径总长度 | | 关键路径曾在 2011 年应用题中考过 |
| | | 4.6.2 | 文字描述：关键路径总长度的现实意义是什么? | | |

| | | | | | |
|---|---|---|---|---|---|
| **查找算法的分析和应用** | 分块查找 | 5.1.1 | 自己设计一个分块查找的例子，不少于15个数据元素，并建立分块查找的索引 | ☐ | “查找算法”部分，在应用题中最有可能专门考察的是 分块查找、折半查找、二叉查找树、平衡二叉树、散列查找。<br><br>其中，二叉查找树、平衡二叉树的可能考法已在本文档的 3.6 有详细介绍。此处我们主要训练 分块查找、折半查找、散列查找在应用题中的可能考法 |
| | | 5.1.2 | 基于上述例子，计算查找成功的ASL、查找失败的ASL | ☐ | |
| | 折半查找 | 5.2.1 | 自己设计一个折半查找的例子，不少于10个数据元素，画出对应的查找分析树 | ☐ | |
| | | 5.2.2 | 基于上述例子，计算查找成功的ASL、查找失败的ASL | ☐ | |
| | 散列查找 | 5.3.1 | 自己设计一个散列表，总长度由你决定，并设计一个合理的散列函数，使用线性探测法解决冲突 | ☐ | |
| | | 5.3.2 | 基于上述散列表，设计不少于10个元素的插入序列，依次插入散列表，画出散列表最终的样子（插入过程至少发生4次冲突） | ☐ | |
| | | 5.3.3 | 基于上述例子，计算查找成功的ASL、查找失败的ASL | ☐ | |
| | | 5.3.4 | 自己设计一个散列表，总长度由你决定，并设计一个合理的散列函数，使用拉链法解决冲突 | ☐ | |
| | | 5.3.5 | 基于上述散列表，设计不少于10个元素的插入序列，依次插入散列表，画出散列表最终的样子（插入过程至少发生4次冲突） | ☐ | |
| | | 5.3.6 | 基于上述例子，计算查找成功的ASL、查找失败的ASL | ☐ | |

| | | | | | |
|---|---|---|---|---|---|
| **排序算法的分析和应用** | 希尔排序 | 6.1.1 | 自己设计一个长度不小于10的乱序数组，用希尔排序，自己设定希尔排序参数 | ☐ | “排序算法”部分，在应用题中最有可能考察的算法，应该具备如下特性：<br>1. 算法的代码比较复杂，不适合考代码<br>2. 算法不能太简单，太简单的算法在选择题中考察即可（如：插入排序、选择排序、冒泡排序）<br><br>因此，在内部排序算法中，应用题部分应该重点关注：希尔排序、堆排序、基数排序。另外，快速排序通常在算法题中考察，不过仍然建议大家画图梳理一遍的快排算法的执行过程。 |
| | | 6.1.2 | 画出每一轮希尔排序的状态 | ☐ | |
| | 堆排序 | 6.2.1 | 自己设计一个长度不小于10的乱序数组，用堆排序，最终要生成升序数组，画出建堆后的状态 | ☐ | |
| | | 6.2.2 | 画出每一轮堆排序的状态 | ☐ | |
| | 快速排序 | 6.3.1 | 自己设计一个长度不小于10的乱序数组，用快速排序，最终要生成升序数组 | ☐ | |
| | | 6.3.2 | 画出每一轮快速排序的状态 | ☐ | |
| | 基数排序 | 6.4.1 | 自己设计一个长度不小于15的乱序链表，每个数据元素取值范围0~99，用基数排序，最终要生成升序链表 | ☐ | |
| | | 6.4.2 | 画出每一轮基数排序的状态 | ☐ | |
| **外部排序专题** | 置换选择排序 | 6.5.1 | 完成《【数据结构应用题】打卡表参考文档》6.5第（1）小问 | ☐ | |
| | 最佳归并树 | 6.5.2 | 完成《【数据结构应用题】打卡表参考文档》6.5第（2）小问 | ☐ | |
| | 败者树 | 6.5.3 | 完成《【数据结构应用题】打卡表参考文档》6.5第（3）小问 | ☐ | |