数据结构与算法 核心知识提要

夏天明

2024 年 6 月 17 日

目录

0 序

笔者参考了闫宏飞老师的课程资料（以及课程群里大家的讨论）和郭炜老师的课件，总结了数据结构与算法课程中一些较为重要的、以及可能容易混淆的知识点，并分类整理了往年笔试题目，供读者参考。

1 基本概念：算法与数据结构

1.1 算法

定义 1.1 (算法). 算法是对计算过程的描述，是为了解决某个问题而设计的**有限长**操作序列。

算法的特性：

- **有穷性：**
  - 只有有限个指令
  - 有限次操作后终止
  - 每次操作在有限时间内完成
  - 终止后必须给出解或宣告无解
- **确定性：**相同输入得到相同输出
- **可行性：**无歧义，可以机械执行
- **输入/输出：**可以不需要输入，但必须有输出

常用的算法思想有枚举、二分、贪心等等，这部分需要注意的是**递归**和**分治**的区别：都是将原问题变成形式相同但规模更小的问题，但递归是通过**先采取一步行动**实现，分治是通过**分解为几个子问题**实现。另外，分治往往用递归实现。
关于动态规划：一个问题必须拥有重叠子问题和最优子结构，才能使用动态规划去解决。

例 1.1. 下列不影响算法时间复杂性的因素有（C）。
A：问题的规模　B：输入值
C：计算结果　D：算法的策略

1.2 数据结构

定义 1.2 (数据结构). 数据结构就是数据的组织和存储形式，是**带有结构的数据元素的集合**。描述一个数据结构，需要指出其**逻辑结构**、**存储结构**和**可进行的操作**。

数据的单位称作**元素**或**节点**。数据的基本单位是数据元素，最小单位是数据项。逻辑结构有：

- **集合结构：**各元素无逻辑关系
- **线性结构：**除了最前、最后的节点外都有前驱和后继
- **树结构：**根节点无前驱，其他节点有 1 个前驱
- **图结构**

存储结构（数据在内存中的存储方式）：

- **顺序结构：**连续存放
- **链接结构：**不连续，存指针指向前驱/后继
- **索引结构：**每个结点有一个关键字，关键字的指针指向该节点
- **散列结构：**根据散列函数计算存储位置

注. 关于散列表需要记忆的是处理冲突的方法：

- **线性探测法：**如果位置`H(x)`被占，则探测`(H(x)+d)%m`，`d=1,2,...`
- **二次探测法：**探测`(H(x)+d)%m`，`d=1,-1,4,-4,...`
- **再散列法：**涉及第二个散列函数`H_2(x)`，探测`(H(x)+d*H_2(x))%m`，`d=1,2,...`

二次聚集现象：非同义词争夺同一个后继位置，即处理同义词冲突的时候又产生了非同义词的冲突

例 1.2. 给定一个长度为 7 的空散列表，采用双散列法解决冲突，两个散列函数分别为：$\mathrm{h1(key)} = \mathrm{key}\ \%\ 7$，$\mathrm{h2(key)} = \mathrm{key}\%5 + 1$ 请向散列表依次插入关键字为 30, 58, 65 的集合元素，插入完成后 65 在散列表中存储地址为（3）。

例 1.3. （N）由于碰撞的发生，基于散列表的检索仍然需要进行关键码对比，并且关键码的比较次数仅取决于选择的散列函数与处理碰撞的方法两个因素。

可进行的操作：建立、插入、删除、查找结点，求结点前驱或结点后继，随机访问。数据的逻辑结构与存储结构无关。

例 1.4. 16. 下列叙述中正确的是（A）。
A：一个逻辑结构可以有多种类型的存储结构，且不同类型的存储结构会直接影响到数据处理的效率
B：散列是一种基于索引的逻辑结构
C：基于顺序表实现的逻辑结构属于线性结构
D：数据结构设计影响算法效率，逻辑结构起到了决定作用

例 1.5. 与数据元素本身的形式、内容、相对位置、个数无关的是数据的（C）。A. 存储结构　B. 存储实现　C. 逻辑结构　D. 运算实现

2　线性表

线性表中的元素属于相同的数据类型，每个元素所占的空间必须相同。**串的长度**定义为串中所含字符的个数。
串是一种特殊的线性表，其数据元素是**一个字符**。

2.1　顺序表

即 Python 中的列表和其他语言中的数组。

例 2.1. 对长度为 3 的顺序表进行查找，若查找第一个元素的概率为 1/2，查找第二个元素的概率为 1/4，查找第三个元素的概率为 1/8，则执行任意查找需要比较元素的平均个数为（　）。
$1*(1/2)+2*(1/4)+3*(1/8)+3*(1/8)=1.75$, 还有 1/8 的失败查询概率。

例 2.2. 对 n 个元素的表做顺序查找时，若查找每个元素的概率相同，则平均查找长度为（$\frac{n+1}{2}$）。不考虑找不到的情形

例 2.3. 向一个有 127 个元素的顺序表中插入一个新元素并保持原来顺序不变，平均要移动的元素个数为（　）　*63.5*

2.2　链表

- **单链表：**每个元素存后继的指针。
- **循环单链表：**单链表尾元素额外存头部的指针。
- **双向链表：**单链表每个元素额外存前驱的指针。
- **循环双向链表**

例 2.4. 若某线性表常用操作是在表尾插入或删除元素，则时间开销最小的存储方式是(顺序表)。各数据结构对应队尾插入/删除复杂度如下：
A：单链表　$O(n)$，$O(n)$
B：仅有头指针的单循环链表　$O(n)$，$O(n)$
C：顺序表　$O(1)$，$O(1)$
D：仅有尾指针的单循环链表　$O(1)$，$\boldsymbol{O(n)}$

例 2.5. 对于单链表，表头节点为`head`，判定空表的条件是（ `head==None` ）。

例 2.6. 单链表的存储密度（C）。
A. 大于 1 B. 等于 1 C. 小于 1 D. 不能确定
单链表还要存储指针

3 栈

后进先出（LIFO）
波兰表达式：即前缀表达式。逆波兰表达式即后缀表达式。

例 3.1. 使用栈计算后缀表达式 (操作数均为一位数) “1 2 3 + 4 * 5 + 3 + −”，当扫描到第二个 + 号但还未对该 + 号进行运算时，栈的内容 (以栈底到栈顶从左往右的顺序书写) 为（　）。*1,20,5*

例 3.2. 若让元素 1，2，3，4，5 依次进栈，则出栈次序不可能出现在（C）种情况。
A. 5，4，3，2，1
B. 2，1，5，4，3
C. 4，3，1，2，5
D. 2，3，5，4，1
每个数后面比它小的数都应为递减排列的。也可以直接手动模拟。

4 队列

先进先出（FIFO）实现方法：

- 用足够大的列表实现，维护队头指针和队尾指针（浪费空间）
- **列表 + 头尾循环法**。队头指针`front`指向第一个元素，队尾指针`rear`指向最后一个元素后面
  判断队列是否空/满：
  - 维护一个变量`size`记录队列中元素总数
  - 不维护`size`，浪费一个单元的存储空间，即`rear`所指单元。则`front==rear`即为空，`front==(rear+1)%capability`即为满
- 用两个栈实现一个队列：从`inStack`压入，`outStack`弹出，若`outStack`空，将`inStack`全部弹出并逐个压入`outStack`。由于每个元素最多进出两个栈一次，故平均时间复杂度为 $O(1)$。

例 4.1. 设环形队列的容量为 20（单元编号从 0 到 19），现经过一系列的入队和出队运算后，队头变量（第一个元素的位置）front=18，队尾变量（待插入元素的位置）rear=11，在这种情况下，环形队列中有（　）个元素。　*13*

例 4.2. 循环队列存储在数组 A[0..m] 中，则入队时的操作为（D ）。
A. `rear=rear+1`　B. `rear=(rear+1)%(m-1)`
C. `rear=(rear+1)%m`　D. `rear=(rear+1)%(m+1)`

5 二叉树

5.1 基本概念

定义 5.1 (二叉树).

1. 二叉树是有限个元素的集合
2. 空集合是一个二叉树，称为**空二叉树**
3. 根 + 左子树 + 右子树能形成一个新的二叉树。**要求根、左子树和右子树三者没有公共元素**。

二叉树的左右子树是有区别的。

定义 5.2 (结点的度). 结点的非空子树数目称为结点的度。

定义 5.3 (结点的深度、层次). 根节点是第 0 层的。结点的深度即为其层次。

定义 5.4 (树的高度). 树的高度被定义为所有节点的最大层次数。

定义 5.5 (特殊的二叉树).

- **完美二叉树（满二叉树）：**每层结点数目达到最大
- **真二叉树：**没有 1 度结点的二叉树
- **完全二叉树：**除最后一层外，其余层结点数目达到最大

二叉树的遍历：层序遍历（广度优先），前/中/后序遍历（深度优先）

例 5.1. （Y）按照前序、中序、后序方式周游一棵二叉树，分别得到不同的结点周游序列，然而三种不同的周游序列中，叶子结点都将以相同的顺序出现。

例 5.2. （Y）通过树的周游可以求得树的高度，若采取深度优先遍历方式设计求解树高度问题的算法，算法**空间**复杂度大 O 表示为 O（树的高度）。 时间复杂度为 $O(n)$

例 5.3. 若定义二叉树中根结点的层数为零，树的高度等于其结点的最大层数加一。则当某二叉树的前序序列和后序序列正好相反，则该二叉树一定是（B）的二叉树。
A：空或只有一个结点 B：高度等于其节点数
C：任一结点无左孩子 D：任一结点无右孩子

例 5.4. 给定一个二叉树，若前序遍历序列与中序遍历序列相同，则二叉树是（只有根结点的二叉树或非叶子结点只有右子树的二叉树）

例 5.5. （N）若有一个叶子结点是二叉树中某个子树的前序遍历结果序列的最后一个结点，则它一定是该子树的中序遍历结果序列的最后一个结点。 例如：根节点是 A，前序 AB，中序 BA

5.2 哈夫曼树

定义 5.6 (哈夫曼树). 给定 n 个结点及其权值 W_i，构造一棵二叉树，其叶子结点是给定的结点，且最小化 $\sum_i W_i \cdot L_i$（其中 L_i 表示叶结点 i 到树根的路径长度），则称为哈夫曼树，又叫**最优二叉树**。

哈夫曼树并不唯一。构造方法：每次取出权值最小的两个节点 i, j，构造新节点 r 将这两个结点作为子节点，并赋予权重 $W_i + W_j$，并放回 r。
哈夫曼编码：采用**前缀编码**，任何一个字符的编码都不会是其他字符编码的前缀。

5.3 堆

定义 5.7 (堆). 堆是一个完全二叉树；堆中任何结点优先级都大于等于其两个子结点。

堆的相关操作：

- **添加元素：** 上移，$O(logN)$
- **删除元素：** 堆顶与最后一个元素交换，然后下移，$O(logN)$
- **从无序列表建堆：** 从倒数第二层开始，逐层向上遍历，对每个非叶子进行下移。$O(N)$

5.4 二叉搜索树

定义 5.8 (二叉搜索树). 每个结点存储关键字 (key) 和值 (value) 两部分数据；对每个结点 X，其左子树中的全部结点的 key 都小于 X 的 key，且右子树中的全部结点的 key 都大于 X 的 key。

一个二叉树是二叉搜索树，**当且仅当**其中序遍历是递增序列。
二叉搜索树的查找和增添操作是显然的。对于删除操作，如果要删除的结点的度 <2，直接删去，把下面的上移即可。如果度 =2，找左子树的最右结点或者右子树的最左结点，把值放到这里，再把那个结点删去（此时回到上一种情况）。

例 5.6. 由同一组关键字集合构造的各棵二叉排序树 (B)。
A：其形态不一定相同，但平均查找长度相同
B：其形态不一定相同，平均查找长度也不一定相同
C：其形态均相同，但平均查找长度不一定相同
D：其形态均相同，平均查找长度也都相同

例 5.7. （Y）有 n 个节点的二叉排序树有多种，其中树高最小的二叉排序树是搜索效率最好。

例 5.8. 在映射抽象数据类型（ADT Map）的不同实现方法中，适合对动态查找表进行高效率查找的组织结构是（C）。
A：有序表　B：堆排序　C：二叉排序树　D：快速排序

注. 有序表是静态的 ADT Map.

6　树与森林

定义 6.1 (树). 每个结点可以有任意多棵不相交的子树；子树有序，从左到右依次是子树 1，子树 2...

二叉树的结点在只有一棵子树的情况下，要区分是左子树还是右子树。树的结点在只有一棵子树的情况下，都算其是第 1 棵子树**（所以二叉树不是树）**。

n 个结点的树有 n-1 条边。

树转化为二叉树：左儿子右兄弟，树的**前序遍历**和儿子兄弟树的**前序遍历**一致，树的**后序遍历**和儿子兄弟树的**中序遍历**一致。得到的二叉树是**唯一的**。

树的储存形式：双亲表示法、孩子链表表示法、孩子兄弟表示法
树的储存结构：顺序储存、链式储存

定义 6.2 (森林). 不相交的树的集合就是森林，且森林有序：第 1 棵树，第 2 棵树...

转换为二叉树：二叉树的根 S1 是第一棵树的根，S1 的右子结点 S2 是第二课树的根...

例 6.1. 考虑一个森林 F，其中每个结点的子结点个数均不超过 2。如果森林 F 中叶子结点的总个数为 L，度数为 2 结点（子结点个数为 2）的总个数为 N，那么当前森林 F 中树的个数为（$L-N$）。

注. 一棵二叉树中，叶节点比二度节点数量多一。

例 6.2. 设 F 是一个森林，B 是由 F 变换得的二叉树。若 F 中有 n 个非终端结点，则 B 中右指针域为空的结点有（C）个。A. n-1　B. n　C. n+1　D. n+2

7　图

7.1　基本概念

无向图两个顶点之间最多有 1 条边；有向图两个顶点之间最多有两条方向相反的边。

定义 7.1 (顶点的相关概念).

- 顶点的度数：所连边数。有向图中还有出度、入度
- 有向图中顶点的出边：从该顶点出发的边；入边：终点为该顶点的边

定义 7.2 (路径的相关概念)**.**

- **路径：**顶点序列，相邻两个点存在边（有向图中则要固定方向）
- **回路（环）：**起点与终点相同的路径
- **简单路径：**除了起点和终点可能相同外，其它顶点都不相同

定义 7.3 (图的相关概念)**.**

- **完全图：**任意两个顶点之间都有一条（无向图）/两条（有向图）边
- **可达：**若存在 u 至 v 的路径，则称 u 可达 v
- **连通无向图/强连通有向图：**任意两个顶点互相可达
- **网络：**带权无向连通图

定义 7.4 (图的表示)**.**

邻接表：每个顶点加一个边表（有向图记录出边）

逆邻接表/邻接入边表：有向图记录入边

相邻矩阵 (A)：是一个 $n \times n$ 的矩阵，A_{ij} 的值为 1 表示 i 和 j 节点之间有边，0 表示没有边。

路径矩阵：相邻矩阵的 m 次幂 A^m 的 i 行 j 列的值表示 i 和 j 节点之间的路径的数量。如果该元素不为零，说明存在长度为 m 的路径。

例 7.1. 令 G=(V, E) 是一个无向图，若 G 中任何两个顶点之间均存在唯一的简单路径相连，则下面说法中错误的是（A）。
A：图 G 中添加任何一条边，不一定造成图包含一个环　和原来存在的简单路径共同构成一个环
B：图 G 中移除任意一条边得到的图均不连通　两个端点之间不再存在简单路径
C：图 G 的逻辑结构实际上退化为树结构　无向连通无环 = 树
D：图 G 中边的数目一定等于顶点数目减 1　树的特性

例 7.2. 有 n（n>=2）个顶点的有向强连通图最少有（　）条边。　有 n 条边，考虑形成一个大环即可

例 7.3. 2. 已知一个无向图 G 含有 18 条边，其中度数为 4 的顶点个数为 3，度数为 3 的顶点个数为 4，其他顶点的度数均小于 3，请问图 G 所含的顶点个数至少是（*13*）。
一条边贡献两个度数

例 7.4. 图的 BFS 生成树的树高比 DFS 生成树的树高（C）。
A. 小　B. 相等　C. 小或相等　D. 大或相等

7.2 拓扑排序

定义 7.5 (拓扑排序)**.** 有向图中，满足以下条件的序列被称为拓扑排序：

1. 每个顶点出现且只出现一次
2. 若存在一条从顶点 A 到顶点 B 的路径，那么在序列中顶点 A 出现在顶点 B 的前面

有向图存在拓扑排序当且仅当其为有向无环图（DAG）
入度法求拓扑排序：$O(V+E)$
AOV 网络：在有向图中，用顶点表示活动，用有向边表示前驱是后驱必要活动条件

7.3 最小生成树

定义 7.6 (最小生成树)**.** n 个顶点的无向带权连通图中，若一包含全部顶点的子图连通且无环，则称其为原图的**生成树**。生成树必然有 n-1 条边。所有边权值和最小的生成树称为**最小生成树**。

求法：

- Prim $O(E\log V)$
- Kruskal $O(E\log E)$

一个图的两棵最小生成树，**边的权值序列**排序后结果相同。

例 7.5. （Y）如果一个连通无向图 G 中所有边的权值均不同，则 G 具有唯一的最小生成树。

例 7.6. （N）对任意一个连通的无向图，如果存在一个环，且这个环中的一条边的权值不小于该环中任意一个其它的边的权值，那么这条边一定不会是该无向图的最小生成树中的边。 正确的说法：如果环中一边严格大于其他所有边的权值，那么必不在最小生成树中。

例 7.7. 51 个顶点的连通图 G 有 50 条边，其中权值为 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 的边各 5 条，则连通图 G 的最小生成树各边的权值之和为（ ）。*275*

7.4 最短路问题

Dijkstra 适用场景：**无负权边**的带权有向图 或 无向图 的**单源**最短路问题。$O(E\log V)$
Floyd 适用场景：求每一对顶点之间的最短路径。有向图，无向图均可。有向图**可以有负权边**，但是**不能有负权回路**。$O(n^3)$

8 排序

- **插入排序：**将列表分为左面有序部分（初始为 0 个元素）和右面无序部分，每次把无序部分的第一个元素插入到有序部分合适的位置（通过**从后向前遍历**）。平均：$O(n^2)$，最坏（倒序）：$O(n^2)$，最好（顺序）：$O(n)$。稳定。
- **希尔排序：**最好 $O(n)$，平均 $O(n^{1.5})$，最坏 $O(n^2)$。不稳定。
- **选择排序：**同样分左右两部分，每次找到无序部分最小的，与无序部分第一位交换。复杂度总是 $O(n^2)$。不稳定。
- **冒泡排序：**最好（顺序）：$O(n)$，平均：$O(n^2)$，最坏（倒序）：$O(n^2)$。稳定。
- **归并排序：**切分 + 合并，总是 $O(n\log n)$。需要额外空间 $O(n)$，需要栈空间 $O(\log n)$。稳定。
- **快速排序：**若使用第一个元素作为 pivot，最坏情况为基本顺序或倒序 $O(n^2)$，最好 & 平均 $O(n\log n)$。需要栈空间 $O(\log n)$，最坏 $O(n)$。不稳定。
- **堆排序：**一种特殊的**选择**排序。先 $O(n)$ 建大顶堆，列表分为左面堆部分和右面有序部分，每次将堆顶与堆末交换，新堆顶重新下沉。故总是 $O(n\log n)$。不稳定。
- **桶排序：**桶！$O(n+m)$，m 为桶数。空间需要 $O(n+m)$，因为每个桶可能对应多个。稳定。
- **多轮分配排序（基数排序）：**相当于 d 次桶排序，因此为 $O(d(n+m))$。稳定。

例 8.1. 排序算法依赖于对元素序列的多趟比较/移动操作（即执行多轮循环），第一趟结束后，任一元素都无法确定其最终排序位置的算法是（D ）。

A：选择排序　最小的被确定

B：快速排序　*pivot* 被确定

C：冒泡排序　最大的被确定

D：插入排序

例 8.2. 如果输入序列是已经正序，在 (改进) 冒泡排序、直接插入排序和直接选择排序算算法中，**直接选择排序**算法最慢结束。

例 8.3. 假设线性表中每个元素有两个数据项 key1 和 key2，现对线性表按以下规则进行排序：先根据数据项 key1 的值进行非递减排序；在 key1 值相同的情况下，再根据数据项 key2 的值进行非递减排序。满足这种要求的排序方法是（D）。

A: 先按 key1 值进行冒泡排序，再按 key2 值进行直接选择排序

B: 先按 key2 值进行冒泡排序，再按 key1 值进行直接选择排序

C: 先按 key1 值进行直接选择排序，再按 key2 值进行冒泡排序

D: 先按 key2 值进行直接选择排序，再按 key1 值进行冒泡排序

只需保证先排 *key2* 再排 *key1*，且第二次是稳定排序即可。

例 8.4. 有 n 个数据对象的二路归并排序中，每趟归并的时间复杂度为（　）。　*O(n)*

例 8.5. 给定一个 N 个相异元素构成的有序数列，设计一个递归算法实现数列的二分查找，考察递归过程中栈的使用情况，递归调用栈的最小容量应为（$\lceil \log_2(N+1) \rceil$）。

例 8.6. 对 n 个不同的排序码进行冒泡排序，在元素无序的情况下比较的次数最多为（$n(n-1)/2$）。

9 字符串匹配

朴素算法： 复杂度 $O(m*n)$
前缀函数： 数组的第 n 项为模式串前 n 个字母的最长的相等真前后缀长度 l，即模式串前 l 个字母和后 l 个字母相同。例如字符串`aabaaab`的前缀函数为 $[0,1,0,1,2,2,3]$。
next 数组： 模式串的前缀函数去掉最后一项，前面再加上一项-1。例如字符串`aabaaab`的 next 数组为 $[-1,0,1,0,1,2,2]$。

KMP 算法的时间复杂度：$O(n+m)$，m 代表模式串长度（因为指针单向移动）。额外空间（next 数组）：$O(m)$。