数据结构复习提纲

> 根据课程重点整理，涵盖各章节核心考点与示例

第 1 章绪论 / 时间复杂度

考点

- while 循环的时间复杂度分析，特别是**基本语义变量与循环变量有关联**的情况

示例问题

问题：分析以下代码的时间复杂度

```
int i = 1;
while (i <= n) {
    i = i * 2;
}
```

答案：$O(\log n)$。循环变量 i 以指数增长，执行次数为 $\log n$。

问题：分析以下嵌套循环的时间复杂度

```
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= i; j++) {
        // 操作
    }
}
```

答案：$O(n^2)$。内层循环次数为 $1+2+3+\ldots+n = n(n+1)/2$。

第 2 章线性表

2.1 顺序表

考点

- 插入元素时**移动元素个数**：分最好情况、最坏情况、平均情况
- 顺序表支持**随机存取**（$O(1)$ 时间访问任意位置）

示例问题 **问题**：在长度为 n 的顺序表中插入一个元素，分别求最好、最坏、平均情况下需要移动的元素个数。

答案：

情况	插入位置	移动元素个数
最好	表尾（第 $n+1$ 位）	0
最坏	表头（第 1 位）	n
平均	等概率随机插入	$n/2$

平均情况推导：等概率插入 n+1 个位置，移动次数分别为 n, n-1, …, 1, 0

$$ASL = \frac{1}{n+1}\sum_{i=0}^{n} i = \frac{n(n+1)/2}{n+1} = \frac{n}{2}$$

2.2 链表（重点）

考点

- **链表原地逆置**：不借助额外空间，将链表元素翻转（空间复杂度 O(1)）

示例问题　问题：给定一个带头结点的单链表，编写算法在不使用额外数据空间的情况下将其逆置。

答案：

```
typedef struct LNode {
    int data;
    struct LNode *next;
} LNode, *LinkList;

// 原地逆置单链表
void Reverse(LinkList &L) {
    LNode *p = L->next;    // p 指向第一个数据节点
    L->next = NULL;        // 头结点指向空，作为新链表的尾

    while (p != NULL) {
        LNode *q = p;      // q 保存当前节点
        p = p->next;       // p 后移
        q->next = L->next; // 头插法：q 插入到头结点之后
        L->next = q;
    }
}
```

解析：使用三个指针（p 遍历原链表，q 保存当前节点，L->next 指向新链表头），逐个将节点头插到新链表中，实现逆置。空间复杂度 O(1)，时间复杂度 O(n)。

第 3 章栈和队列

3.1 栈

考点

- 给定入栈序列，判断**可能的出栈次序**
- 栈的本质是**操作受限的线性表**（先进后出 FILO）

示例问题　问题：入栈序列为 1, 2, 3，判断以下出栈序列是否可能：

- (1) 3, 2, 1
- (2) 2, 3, 1
- (3) 3, 1, 2

答案:

- (1) **可能**。1 入 →2 入 →3 入 →3 出 →2 出 →1 出
- (2) **可能**。1 入 →2 入 →2 出 →3 入 →3 出 →1 出
- (3) **不可能**。3 先出说明 1,2,3 都已入栈，此时栈顶是 2，1 在栈底，1 不可能在 2 之前出栈。

3.2 队列

考点

- **循环队列的长度计算**：区分长度（元素个数）与容量
- **循环队列的队空和队满条件**
- 队列也是受限线性表（先进先出 FIFO）

示例问题　问题：循环队列用数组实现，队头指针 front，队尾指针 rear，数组容量为 MaxSize。如何计算队列长度？队空和队满的条件是什么？

答案:

```
// 牺牲一个单元区分队空和队满
队列长度 = (rear - front + MaxSize) % MaxSize;

// 队空条件：front == rear
// 队满条件：(rear + 1) % MaxSize == front
```

解析：循环队列通过取模运算实现”循环”，长度计算需考虑 rear 可能在 front 之前的情况。牺牲一个存储单元是为了区分队空和队满两种状态。

第 4 章串

4.1 数组

考点

- **数组元素地址计算**：给定行列数、元素大小、按行/列优先存储方式
- **二维数组的地址计算**

示例问题　问题：二维数组 A[0..9][0..9]，每个元素占 4 个存储单元，按行优先存储，基地址为 1000。求 A[2][5] 的地址。

答案:

$$LOC(A[i][j]) = LOC(A[0][0]) + (i \times n + j) \times L$$

$$LOC(A[2][5]) = 1000 + (2 \times 10 + 5) \times 4 = 1000 + 25 \times 4 = 1100$$

4.2 模式匹配

考点

- **KMP 算法**相对于 BF 算法的**主要优势**

示例问题　问题：KMP 算法相对于 BF（暴力）算法的主要优势是什么?

答案:

1. **主串指针不回溯**：BF 算法匹配失败时主串指针回溯，KMP 利用已匹配信息避免回溯
2. **时间复杂度 $\mathbf{O(m+n)}$**：BF 为 $O(m \times n)$，KMP 为线性时间
3. **通过 next 数组**：预处理模式串，利用最长公共前后缀信息跳过不必要的比较

4.3 广义表

考点

- **广义表的深度**：计算括号嵌套层数

示例问题　问题：求广义表 `A = (a, (b, c), ((d, e), f))` 的深度。

答案：**深度为 3**。

解析：深度 = 括号的最大嵌套层数。

- 第 1 层：A 本身
- 第 2 层：(b, c) 和 ((d, e), f)
- 第 3 层：(d, e)

第 5 章树和二叉树

5.1 完全二叉树性质

考点

- **完全二叉树叶子节点数**：给定 n 个节点，会算叶子节点数
- 二叉树性质：**度为 1 的节点要么为 0，要么为 1**
- **满二叉树 vs 完全二叉树**：满二叉树一定是完全二叉树，但完全二叉树不一定是满二叉树

示例问题　问题 1：一棵完全二叉树有 100 个节点，求叶子节点数。

答案:

- 设 n 为叶子节点数，n 为度为 1 的节点数，n 为度为 2 的节点数
- 节点总数：$n = n + n + n = 100$
- 分支数：$n - 1 = n + 2n$（总分支数 = 总节点数 - 1）
- 联立得：$n = n + 1$
- 完全二叉树中 n 只能是 0 或 1
- 代入：$100 = n + n + (n - 1) = 2n + n - 1$
- 若 $n = 0$：$2n = 101$（不成立，n 不是整数）
- 若 $n = 1$：$2n = 100$，$\mathbf{n = 50}$

叶子节点数为 50

问题 2：完全二叉树中度为 1 的节点个数有什么规律?

答案:

- 完全二叉树中度为 1 的节点个数**只能是 0 或 1**
- 原因：完全二叉树只在最底层从右往左连续缺少若干节点
- 若节点总数 n 为**偶数**：度为 1 的节点数为 **1**
- 若节点总数 n 为**奇数**：度为 1 的节点数为 **0**

5.2 遍历序列恢复二叉树

考点

- 由**先序 + 中序**或**后序 + 中序**恢复二叉树

示例问题　问题：先序序列为 ABDECF，中序序列为 DBEACF，画出该二叉树。

答案:

步骤:

1. 先序第一个元素 **A** 是根节点
2. 在中序中找到 A，左边 **DBE** 是左子树，右边 **CF** 是右子树
3. 左子树先序为 BDE，中序为 DBE → B 是根，D 在左，E 在右
4. 右子树先序为 CF，中序为 CF → C 是根，F 在右

```
      A
     / \
    B   C
   / \    \
  D   E    F
```

5.3 线索二叉树

考点

- 根据遍历序列画**先序线索二叉树**

示例问题　问题：画出上述二叉树的先序线索二叉树。

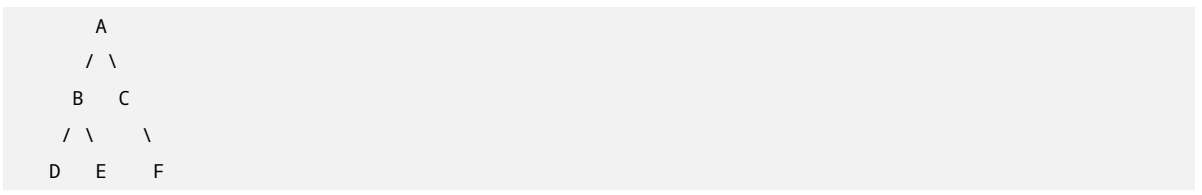
答案:

先序遍历序列：$\mathbf{A} \rightarrow \mathbf{B} \rightarrow \mathbf{D} \rightarrow \mathbf{E} \rightarrow \mathbf{C} \rightarrow \mathbf{F}$

线索化规则:

- 若左孩子为空，则左指针指向其先序前驱

- 若右孩子为空，则右指针指向其先序后继



线索：

- D 的左线索为空（先序第一个），右线索指向 E
- E 的左线索为空，右线索指向 C
- F 的左线索为空，右线索为空（先序最后一个）

5.4 树、森林与二叉树的转换

考点

- **树　二叉树　森林**的相互转换（孩子兄弟表示法）

示例问题　问题：将以下森林转换为二叉树。

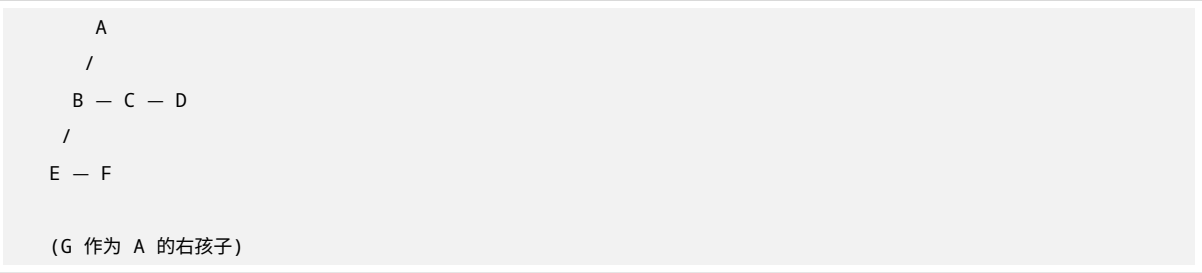
森林：

- 树 1：A(B(E,F), C, D)
- 树 2：G(H, I)

答案：

步骤：

1. 每棵树先转为二叉树（左孩子右兄弟）
2. 树 1 转二叉树：A 为根，B 为左孩子，C 为 B 的右兄弟，D 为 C 的右兄弟；E、F 为 B 的左孩子和右兄弟
3. 树 2 转二叉树：G 为根，H 为左孩子，I 为 H 的右兄弟
4. 森林转二叉树：将树 2 的根 G 作为树 1 根 A 的右孩子



第 6 章图

6.1 握手定理

考点

- 给定边数和顶点度范围，求顶点数

示例问题　问题：无向图有 18 条边，度为 4 的顶点有 2 个，度为 3 的顶点有 4 个，度为 5 的顶点有 1 个，其余顶点度为 1 或 2。求总顶点数。

答案：

- 握手定理：所有顶点度数之和 $= 2 \times$ 边数 $= 36$
- 已知顶点度数和：$4\times2 + 3\times4 + 5\times1 = 8 + 12 + 5 = 25$
- 剩余度数：36 - 25 = 11
- 设度为 1 的顶点有 x 个，度为 2 的顶点有 y 个
- 方程：$x + 2y = 11$，且 $x + y =$ 剩余顶点数
- 可能解：y=5, x=1（顶点数 6）或 y=4, x=3（顶点数 7）等
- 需根据题目其他条件确定唯一解

6.2 图的遍历

考点

- 会写 **DFS** 和 **BFS** 遍历序列
- 实现区别：DFS 用**栈/递归**，BFS 用**队列**

示例问题　问题：给定下图，从顶点 A 出发写出 DFS 和 BFS 序列。

答案：

- **DFS（深度优先）**：A → B → C → F → E → D（假设邻接表按字母序）
- **BFS（广度优先）**：A → B → D → C → E → F

实现区别：

	DFS	BFS
数据结构	栈（递归隐式栈）	队列
访问顺序	深入到底再回溯	逐层扩展
应用	连通分量、拓扑排序	最短路径（无权图）

6.3 最小生成树

考点

- **Prim 算法**和 **Kruskal 算法**的构造过程

示例问题　问题：用 Prim 算法求以下图的最小生成树。

```
A —2— B —3— C
|4    |1    |5
D —6— E —2— F
```

答案：

Prim 算法步骤（从 A 开始）：

1. 选 A，邻接边：(A,B,2), (A,D,4)，选最小 **(A,B,2)**
2. 集合 {A,B}，邻接边：(A,D,4), (B,E,1), (B,C,3)，选最小 **(B,E,1)**
3. 集合 {A,B,E}，邻接边：(A,D,4), (B,C,3), (E,D,6), (E,F,2)，选最小 **(E,F,2)**
4. 集合 {A,B,E,F}，邻接边：(A,D,4), (B,C,3), (E,D,6)，选最小 **(B,C,3)**
5. 集合 {A,B,E,F,C}，邻接边：(A,D,4), (E,D,6)，选最小 **(A,D,4)**

最小生成树总权值：$2 + 1 + 2 + 3 + 4 = \mathbf{12}$

6.4 拓扑排序

考点

- 适用于**有向无环图（DAG）**
- 有**环图**无法进行拓扑排序
- 拓扑排序应用：**项目管理**（AOV 网）

示例问题　问题：判断以下有向图能否进行拓扑排序。

```
A → B → C
↑       ↓
└———————D
```

答案：

- 存在环：$A \to B \to C \to D \to A$
- **不能进行拓扑排序**
- 原因：环中每个顶点都有前驱，找不到入度为 0 的顶点作为起点

6.5 邻接矩阵

考点

- **无向图的邻接矩阵是对称的**
- 根据**邻接矩阵画图**
- 根据**边集和顶点集**写出邻接矩阵

示例问题　问题：无向图的邻接矩阵有什么特点?

答案:

- **对称矩阵**：A[i][j] = A[j][i]
- 对角线元素为 0（无自环）
- 第 i 行（或第 i 列）非零元素个数 = 顶点 i 的度

第 7 章查找

7.1 哈希表（重点）

考点

- 用**除留余数法**构造哈希表
- **三种冲突解决方法**：线性探测法、二次探测法、链地址法
- **线性探测法的缺点**：聚集现象
- **二次探测法**解决冲突时产生的问题
- 哈希表三层结构：下标、关键字位置、查找比较次数
- **平均查找长度（ASL）**计算
- 哈希表长度与地址空间的关系

示例问题　问题 1：关键字序列 {7, 8, 30, 11, 18, 9, 14}，哈希函数 H(key) = key mod 7，用**线性探测法**解决冲突，构造哈希表并计算 ASL。

答案:

下标	0	1	2	3	4	5	6
关键字	7	14	9	8	18	30	11
比较次数	1	7	2	1	2	1	1

计算过程:

- H(7) = 0，下标 0 空，放入，比较 **1** 次
- H(14) = 0，冲突 →1,2,3,4,5,6 空，放入，比较 **7** 次
- H(30) = 2，下标 2 空，放入，比较 **1** 次
- H(11) = 4，下标 4 空，放入，比较 **1** 次
- H(18) = 4，冲突 →5 空，放入，比较 **2** 次
- H(9) = 2，冲突 →3 空，放入，比较 **2** 次
- H(8) = 1，下标 1 空，放入，比较 **1** 次

ASL（成功） = (1+7+1+1+2+2+1) / 7 = **15/7　2.14**

线性探测法缺点：产生”聚集”现象，如 14 需要探测多次才能找到空位。

问题 2：哈希表地址空间为 0~8，哈希表长度是多少?

答案：**9**（0~8 共 9 个地址单元）

解析：哈希表长度 m = 9，哈希函数 H(key) = key mod m 中的 m 就是表长。表长与解决冲突时的增量计算直接相关。

问题 3：二次探测法解决冲突时会产生什么问题?

答案：

- 二次探测法：H = (H(key) + d) mod m，其中 d = 1^2, -1^2, 2^2, -2^2, ...
- **优点**：可以**缓解聚集现象**（避免线性探测的连续堆积）
- **缺点**：可能产生**二次聚集**（二次堆积），且不一定能探测到所有空位

7.2 二叉排序树（BST）

考点

- 给定序列**建立二叉排序树**
- 计算**查找成功时的平均查找长度**

示例问题　问题：给定关键字序列 {50, 30, 70, 20, 40, 60, 80}，建立二叉排序树并计算查找成功时的 ASL。

答案：

```
      50
     /  \
   30    70
  / \   / \
20  40 60  80
```

查找过程：

关键字	所在层	比较次数
50	1	1
30	2	2
70	2	2
20	3	3
40	3	3
60	3	3
80	3	3

ASL（成功） = (1×1 + 2×2 + 4×3) / 7 = **17/7　2.43**

解析：查找时先与根比较，小则走左子树，大则走右子树，递归进行。

7.3 折半查找

考点

- 对有序顺序表进行二分查找
- **查找失败**时最多比较次数

示例问题 **问题**：在有序表 $\{5, 13, 19, 21, 37, 56, 64, 75, 80, 88, 92\}$ 中查找 20，最多比较几次？画出判定树。

答案：

判定树：

```
          56(5)
         /     \
      19(2)    80(8)
     /    \    /    \
  13(1)  37(4) 64(7)  88(10)
 /       /       \      /   \
5(0)   21(3)    75(6)  92(11)
```

查找 20：

1. 与 56 比较，$20 < 56$，进入左子树
2. 与 19 比较，$20 > 19$，进入右子树
3. 与 37 比较，$20 < 37$，进入左子树
4. 与 21 比较，$20 < 21$，应到左子树但为空

查找失败，比较 4 次

最多比较次数 $=$ 树高 $= \log\,(\mathrm{n}{+}1) = \log 12 = \mathbf{4}$

第 8 章排序

8.1 快速排序

考点

- **快速排序过程**
- 快速排序在**数据基本有序时效率最差**

示例问题 **问题**：对序列 $\{49, 38, 65, 97, 76, 13, 27, 49\}$ 进行快速排序，写出第一趟结果。

答案：

选第一个元素 49 为枢轴：

初始：49, 38, 65, 97, 76, 13, 27, 49

过程（双向扫描）：

1. 从右向左找 < 49 的：27，与 49 交换位置 $\rightarrow$ 27, 38, 65, 97, 76, 13, 49, 49
2. 从左向右找 > 49 的：65，与 49 交换 $\rightarrow$ 27, 38, 49, 97, 76, 13, 65, 49
3. 从右向左找 < 49 的：13，交换 $\rightarrow$ 27, 38, 13, 97, 76, 49, 65, 49

4. 从左向右找 >49 的：97，交换 $\rightarrow$ 27, 38, 13, 49, 76, 97, 65, 49
5. 从右向左找 <49 的：无（左右指针相遇）

第一趟结果：**27, 38, 13, 49, 76, 97, 65, 49**（49 已归位，左边都小于 49，右边都大于等于 49）

问题：快速排序在什么情况下效率最差？为什么？

答案：

- **数据基本有序时效率最差**
- 原因：每次选择的枢轴都是当前子序列的最大/最小值，导致划分极不平衡
- 时间复杂度退化为 $\mathbf{O(n^2)}$
- 改进：随机选择枢轴或三数取中法

8.2 堆排序

考点

- **建堆过程**（大根堆/小根堆）
- 堆排序是**增序还是降序**?

示例问题　问题：堆排序如果要排**增序**，应该建大根堆还是小根堆?

答案：

- **排增序 $\rightarrow$ 建大根堆**
- 原因：每次将堆顶（最大值）与最后一个元素交换，最大值放到末尾，然后对前面元素重新调整堆
- 类似地，排降序 $\rightarrow$ 建小根堆

排序算法对比表

算法	最好时间	平均时间	最坏时间	空间复杂度	稳定性
直接插入排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	✅ 稳定
冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	✅ 稳定
快速排序	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$	❌ 不稳定
简单选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	❌ 不稳定
堆排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	❌ 不稳定
归并排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	✅ 稳定

附加：哈夫曼树

考点

- 根据字符频率**构造哈夫曼树**

- 计算**带权路径长度（WPL）**
- 写出**哈夫曼编码**
- 由 n 个权值构成的哈夫曼树有多少个节点
- 根据哈夫曼编码计算报文长度

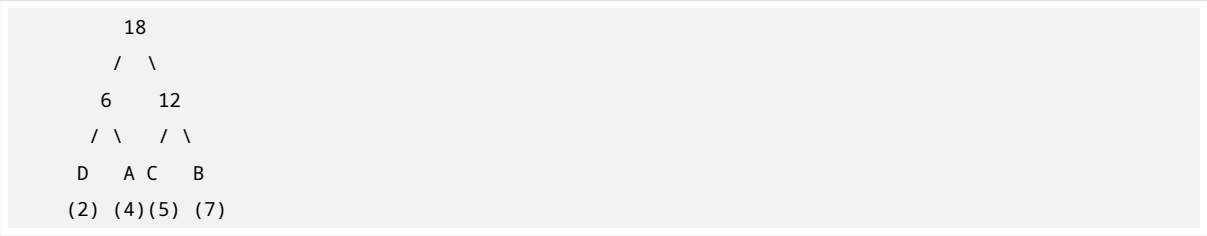
示例问题

问题 1：字符集 $\{A, B, C, D\}$，频率分别为 $\{4, 7, 5, 2\}$，构造哈夫曼树并求 WPL 和编码。

答案：

构造过程：

1. 选最小两个：$D(2)$, $A(4)$ → 合并为 6
2. 选最小两个：$C(5)$, $B(7)$ → 合并为 12
3. 选最小两个：6, 12 → 合并为 18（根）



哈夫曼编码（左 0 右 1）：

字符	编码	路径长度
D	00	2
A	01	2
C	10	2
B	11	2

WPL $= 2\times2 + 4\times2 + 5\times2 + 7\times2 = \mathbf{36}$

或：WPL = 所有非叶子节点权值之和 $= 6 + 12 + 18 = \mathbf{36}$

特点：哈夫曼编码是前缀编码，任何编码都不是另一个编码的前缀，确保解码唯一。

问题 2：由 n 个权值构成的哈夫曼树有多少个节点?

答案：**2n - 1 个节点**

推导：

- n 个叶子节点（权值节点）
- 每次合并产生一个新节点，共需 n-1 次合并
- 总节点数 $=$ n + (n-1) $=$ **2n - 1**

问题 3：根据上述哈夫曼编码，若报文为“ABCD”，计算编码后的报文长度。

答案：

- A: 01 (2 位)
- B: 11 (2 位)
- C: 10 (2 位)
- D: 00 (2 位)
- 报文“ABCD”编码：01 11 10 00
- **总长度 $= 2+2+2+2 = 8$ 位**

答题注意事项

1. **排序题**：不要只写一轮结果，要写出完整的排序过程，每轮都要展示
2. **画图题**：二叉树、线索二叉树、哈夫曼树、图等要画清楚
3. **计算题**：写出推导过程，不要只写结果
4. **算法题**：链表逆置等要写完整代码或清晰步骤

> 祝考试顺利！🎓