计算机 408 统考真题解析

数据结构：综合应用题

408 备考组

2026 年 1 月 2 日

知识模块

1. 线性结构 (11)
2. 树 (4)
3. 图 (10)
4. 排序 + 外排序 (5)
5. 查找 + 高级查找 (6)

大题分布

1. 线性结构 (11)
   - 顺序表 (5)
   - 链表 (4)
   - 栈 (1)
   - 队列 (1)
2. 树 (4)
3. 图 (10)
4. 排序 + 外排序 (5)
5. 查找 + 高级查找 (6)

历年真题

2025 年 408 统考真题 - 算法题 (41)

第 41 题

设有两个长度均为 n 的一维整型数组 A 和 res，对数组 A 中的每个元素 $A[i]$，计算 $A[i]$ 与 $A[j](0 \le i \le j \le n-1)$ 乘积的最大值，并将其保存到 $res[i]$ 中。
示例：若 $A = \{1, 4, -9, 6\}$，则得到 $res = \{6, 24, 81, 36\}$。
请设计一个时间和空间上尽可能高效的算法 `calMulMax`。

核心考点

后缀最大值与最小值的动态维护（处理负数乘积）。

算法设计思想

基本设计思想：
乘积的最大值可能由以下三种情况产生：1. $A[i] > 0$ 时，应乘以其后方（含自身）的最大值；2. $A[i] < 0$ 时，应乘以其后方（含自身）的最小值（负负得正）；3. $A[i] = 0$ 时，结果必为 0。
具体步骤：

1. 从后往前遍历数组 A。
2. 使用两个变量 `maxVal` 和 `minVal` 记录从 $n-1$ 到 i 范围内的最大值和最小值。
3. 计算 $res[i] = \max(A[i] \times \text{maxVal}, A[i] \times \text{minVal})$。
4. 更新 `maxVal` 和 `minVal` 供下一步使用。

复杂度：时间复杂度 $O(n)$，空间复杂度 $O(1)$（不计结果数组）。

C/C++ 代码实现

```
void calMulMax(int A[], int res[], int n) {
    if (n <= 0) return;

    int maxVal = A[n-1]; // 记录后缀最大值
    int minVal = A[n-1]; // 记录后缀最小值

    // 从后向前遍历，保证空间复杂度 O(1)
    for (int i = n - 1; i >= 0; i--) {
        // 更新当前的后缀最值
        if (A[i] > maxVal) maxVal = A[i];
        if (A[i] < minVal) minVal = A[i];

        // 计算可能的两个乘积值
        long long prod1 = (long long)A[i] * maxVal;
        long long prod2 = (long long)A[i] * minVal;

        // 取较大者保存
        res[i] = (int)(prod1 > prod2 ? prod1 : prod2);
    }
}
```

2020 年 408 统考真题 - 算法题 (41)

第 41 题

定义三元组 (a, b, c) 的距离 $D = |a - b| + |b - c| + |c - a|$。给定 3 个升序存储的正整数集合 S_1, S_2, S_3。设计算法计算所有可能三元组中的最小距离。

示例：

$S_1 = \{-1, 0, 9\}, S_2 = \{-25, -10, 10, 11\}, S_3 = \{2, 9, 17, 30, 41\}$

最小距离为 2，相应三元组为 $(9, 10, 9)$。

公式简化提示

若 $a \le b \le c$，则 $D = (b - a) + (c - b) + (c - a) = 2(c - a)$。

即：距离 D 为三元组中最大值与最小值之差的两倍。

算法设计思想

基本设计思想：
采用“多路归并”式的三指针法。

1. 初始时，$i = j = k = 0$，分别指向三个数组的首元素。
2. 循环执行以下步骤：
   - 计算当前三元组 $(S_1[i], S_2[j], S_3[k])$ 的距离 D；
   - 若 D 小于当前最小值，更新 `minD`；
   - **核心逻辑**：找到当前三元组中的最小值（假设为 $S_1[i]$），令其指针 i 后移。
3. 直到任一数组遍历结束。

为什么移动最小值？
距离由最大值与最小值之差决定。只有增大最小值，才可能使差距缩小；增大最大值或中间值只会让差距保持或变大。
复杂度：时间复杂度 $O(n_1 + n_2 + n_3)$，空间复杂度 $O(1)$。

C/C++ 代码实现

```c
#define ABS(x) ((x) < 0 ? -(x) : (x))

int findMinDistance(int S1[], int n1, int S2[], int n2, int S3[], int n3) {
    int i = 0, j = 0, k = 0, minD = 0x7fffffff;
    while (i < n1 && j < n2 && k < n3) {
        // 计算当前距离 D = |a-b| + |b-c| + |c-a|
        int D = ABS(S1[i] - S2[j]) + ABS(S2[j] - S3[k]) + ABS(S3[k] - S1[i]);
        if (D < minD) minD = D;

        // 移动最小值对应的指针
        if (S1[i] <= S2[j] && S1[i] <= S3[k]) i++;
        else if (S2[j] <= S1[i] && S2[j] <= S3[k]) j++;
        else k++;
    }
    return minD;
}
```

考场提醒：1. 必须使用三指针同步移动，暴力 $O(n^3)$ 只能拿部分分；2. 注意三项绝对值相加的逻辑。

2018 年 408 统考真题 - 算法题 (41)

第 41 题

给定一个含 $n(n \geq 1)$ 个整数的数组，设计一个在时间上尽可能高效的算法，找出数组中未出现的最小正整数。

示例：

- 数组 $\{-5, 3, 2, 3\}$ 中未出现的最小正整数是 **1**。
- 数组 $\{1, 2, 3\}$ 中未出现的最小正整数是 **4**。

关键发现

在一个长度为 n 的数组中，未出现的最小正整数一定在 $[1, n+1]$ 范围内。

算法设计思想

基本设计思想：
采用“空间换时间”的策略，利用一个辅助数组标记正整数的出现情况。

1. 建立一个长度为 n 的辅助数组 B，初始化为 0。
2. 遍历原数组 A：
   - 若 $0 < A[i] \leq n$，则在辅助数组对应位置记录 $B[A[i]-1]=1$。
   - 否则（负数、0 或大于 n 的数）直接忽略，因为它们不影响结果。
3. 遍历辅助数组 B，第一个值为 0 的下标 i 对应的 $i+1$ 即为缺失的最小正整数。
4. 若遍历完 B 全为 1，则结果为 $n+1$。

复杂度分析：

- **时间复杂度**：$O(n)$。共需两次遍历。
- **空间复杂度**：$O(n)$。需要一个长度为 n 的辅助数组。

```c
int findMissMin(int A[], int n) {
    int i, *B;
    B = (int *)malloc(sizeof(int) * n); // 分配辅助空间
    // 初始化辅助数组
    for (i = 0; i < n; i++) B[i] = 0;
    // 标记出现过的正整数
    for (i = 0; i < n; i++)
        if (A[i] > 0 && A[i] <= n)
            B[A[i] - 1] = 1;
    // 寻找第一个未出现的正整数
    for (i = 0; i < n; i++)
        if (B[i] == 0) break;
    int result = i + 1;
    free(B); // 释放内存
    return result;
}
```

考场策略

虽然此题有 $O(1)$ 空间复杂度的置换算法，但在 408 考场上，使用 $O(n)$ 空间的辅助数组法逻辑最清晰、最容易写对且同样能拿到时间复杂度满分。

2013 年 408 统考真题 - 算法题 (41)

第 41 题

已知整数序列 $A = (a_0, a_1, \ldots, a_{n-1})$，其中 $0 \leq a_i < n$。若某个元素 x 出现的次数 $m > n/2$，则称 x 为主元素。

任务：设计一个尽可能高效的算法，找出 A 的主元素。若存在则输出该元素，否则输出 -1。

示例：

- $A = (0, 5, 5, 3, 5, 7, 5, 5) \rightarrow$ 输出 **5**。
- $A = (0, 5, 5, 3, 5, 1, 5, 7) \rightarrow$ 输出 **-1**。

算法目标

寻找时间复杂度 $O(n)$，空间复杂度 $O(1)$ 的最优解。

算法设计思想

基本设计思想：摩尔投票法
算法分为“候选人选拔”和“确认”两个阶段：

1. **选拔阶段**：遍历数组，记录一个候选人 c 和计数器 count。
   - 若 count == 0，将当前元素设为 c，count = 1；
   - 若当前元素等于 c，则 count++；否则 count--。

   原理：若存在主元素，其出现次数比其他所有元素之和还多，最后剩下的候选人必是它。
2. **确认阶段**：再次遍历数组，统计候选人 c 出现的实际次数。
3. **输出**：若实际次数 $> n/2$，则输出 c；否则输出 -1。

复杂度分析：

- **时间复杂度**：$O(n)$。共需两次扫描数组。
- **空间复杂度**：$O(1)$。仅需常数个辅助变量。

```
int Majority(int A[], int n) {
    int candidate = A[0], count = 1; // 阶段1: 寻找候选主元素
    for (int i = 1; i < n; i++) {
        if (A[i] == candidate) count++;
        else {
            if (count > 0) count--;
            else {
                candidate = A[i];
                count = 1;
            }
        }
    }
    int actualCount = 0; // 阶段2: 统计候选元素实际出现的次数
    for (int i = 0; i < n; i++)
        if (A[i] == candidate) actualCount++;
    if (actualCount > n / 2) return candidate;
    else return -1;
}
```

易错点拨

注意：第一阶段剩下的候选人并不一定是主元素（如序列 $\{1,2,3\}$ 剩下 3），因此必须进行第二阶段的计数校验。

2010 年 408 统考真题 - 算法题 (42)

第 42 题

设将 $n(n>1)$ 个整数存放到一维数组 R 中。设计一个在时间和空间两方面都尽可能高效的算法，将 R 中序列循环左移 $p(0<p<n)$ 个位置。

变换过程：

由 $\langle x_0, x_1, \ldots, x_{n-1}\rangle$ 变换为 $\langle x_p, x_{p+1}, \ldots, x_{n-1}, x_0, x_1, \ldots, x_{p-1}\rangle$。

示例说明

若 $R=\{1,2,3,4,5,6\}$, $p=2$：

左移后结果为 $\{3,4,5,6,1,2\}$。

算法设计思想

基本设计思想：三步翻转法

可将数组 R 视为由两个子向量组成：$A = \langle x_0, \ldots, x_{p-1} \rangle$ 和 $B = \langle x_p, \ldots, x_{n-1} \rangle$。题目要求由 AB 变为 BA。

1. **第一步**：逆置子向量 A，得到 $A^{-1}B$。
2. **第二步**：逆置子向量 B，得到 $A^{-1}B^{-1}$。
3. **第三步**：逆置整个向量 $(A^{-1}B^{-1})^{-1}$，最终得到 BA。

例如：$\{1,2|3,4,5,6\} \xrightarrow{A} \{2,1|3,4,5,6\} \xrightarrow{B} \{2,1|6,5,4,3\} \rightarrow \{3,4,5,6,1,2\}$ **复杂度分析**：

- **时间复杂度**：$O(n)$。所有元素总共被访问两次（逆置子段一次，全逆置一次）。
- **空间复杂度**：$O(1)$。仅需一个辅助变量用于元素交换。

```
// 实现对数组 R 中从 left 到 right 范围内的元素逆置
void Reverse(int R[], int left, int right) {
    while (left < right) {
        int temp = R[left];
        R[left]  = R[right];
        R[right] = temp;
        left++, right--;
    }
}
void LeftShift(int R[], int n, int p) {
    if (p <= 0 || p >= n) return;
    Reverse(R, 0, p - 1);      // 1. 逆置前 p 个元素 (A)
    Reverse(R, p, n - 1);      // 2. 逆置后 n-p 个元素 (B)
    Reverse(R, 0, n - 1);      // 3. 逆置整个数组 (A'B')' = BA
}
```

考场提分点

1. **通用性**：封装一个 Reverse 函数可以极大简化代码并减少出错率。
2. **参数检查**：在主函数中检查 p 的范围体现了良好的编程习惯。

历年真题

1. 线性结构 (11)
   - 顺序表 (5)
   - 链表 (4)
     - 2019-41
     - 2015-41
     - 2012-42
     - 2009-42
   - 栈 (1)
   - 队列 (1)
2. 树 (4)
3. 图 (10)
4. 排序 + 外排序 (5)

2019 年 408 统考真题 - 算法题 (41)

第 41 题

设线性表 $L=(a_1,a_2,\ldots,a_n)$ 采用带头结点的单链表保存。设计一个空间复杂度为 $O(1)$ 且高效的算法，重新排列 L 为 $L'=(a_1,a_n,a_2,a_{n-1},a_3,a_{n-2},\ldots)$。

结点定义：

```
typedef struct node {
    int data;
    struct node *next;
} NODE;
```

示例分析

原链表：$1\to2\to3\to4\to5\to6$

目标链表：$1\to6\to2\to5\to3\to4$

算法设计思想

基本设计思想：
观察目标链表发现，它是将原链表的后半部分逆置后，交叉插入到前半部分中。

1. **寻找中间结点**：使用快慢指针（`slow` 每次走一步，`fast` 每次走两步）。当快指针到达末尾时，慢指针恰好指向中间结点。将链表一分为二。
2. **逆置后半段**：将后半段链表的所有结点进行原地逆置（头插法或三指针翻转）。
3. **交叉合并**：从前半段和逆置后的后半段依次交替取出结点，拼接成最终链表。

复杂度分析：

- **时间复杂度**：$O(n)$。找中点 $O(n/2)$，逆置 $O(n/2)$，合并 $O(n/2)$。
- **空间复杂度**：$O(1)$。仅使用了若干辅助指针。

```c
void rearrange(NODE *h) {
    if (!h->next || !h->next->next) return;
    NODE *p = h, *q = h, *r, *s;
    // 1. 快慢指针找中点
    while (q->next) {
        p = p->next; q = q->next;
        if (q->next) q = q->next;
    }
    q = p->next; p->next = NULL; // 断开链表
    // 2. 逆置后半段 q（头插法）
    while (q) {
        r = q->next;
        q->next = p->next; p->next = q;
        q = r;
    }
    // 3. 交叉合并
    s = h->next; q = p->next; p->next = NULL;
    while (q) {
        r = q->next;
        q->next = s->next;
        s->next = q;
        s = q->next; q = r;
    }
}
```

关键点

注意：在合并前必须正确断开前半段和后半段的连接，否则会形成环。快慢指针法是处理链表长度不确定的最优解。

2015 年 408 统考真题 - 算法题 (41)

第 41 题

用单链表保存 m 个整数，结点结构为 $\boxed{\text{data}|\text{next}}$，且 $|data| \leq n$。设计一个时间尽可能高效的算法，对于链表中 $data$ 绝对值相等的结点，仅保留第一次出现的结点。

示例：

输入：`HEAD -> 21 -> -15 -> -15 -> -7 -> 15`

输出：`HEAD -> 21 -> -15 -> -7`

核心突破口

已知 $|data| \leq n$，提示我们可以利用空间换时间，建立一个长度为 $n+1$ 的辅助数组。

基本设计思想：
使用一个长度为 $n+1$ 的辅助数组 `attr` 记录已出现的绝对值。

1. 初始化辅助数组所有元素为 0。
2. 从头至尾遍历链表，对于每个结点，计算其数据域的绝对值 $t = |data|$。
3. 检查 `attr[t]`：
   - 若为 0，表示该绝对值首次出现。置 `attr[t] = 1`，继续处理下一个结点。
   - 若为 1，表示该绝对值已出现过。删除当前结点，释放空间。

复杂度分析：时间复杂度 $O(m)$，空间复杂度 $O(n)$。

```c
void removeDuplicates(NODE *h, int n) {
    NODE *p = h->link, *pre = h;
    int *attr = (int *)malloc(sizeof(int) * (n + 1)); // 申请辅助空间
    for (int i = 0; i <= n; i++) attr[i] = 0;          // 初始化
    while (p != NULL) {
        int t = p->data > 0 ? p->data : -p->data; // 取绝对值
        if (attr[t] == 0) {                       // 首次出现
            attr[t] = 1;
            pre = p;                              // 指针前移
            p = p->link;
        } else {                                  // 重复出现
            pre->link = p->link;                  // 删除结点
            free(p);
            p = pre->link;
        }
    }
    free(attr); // 释放辅助数组
}
```

考点回顾

1. **单链表删除**：必须维护一个 pre 指针指向当前结点的前驱，才能完成删除操作。
2. **空间开销**：虽然 $O(n)$ 空间在 n 很大时有压力，但相比于 $O(m^2)$ 的暴力查找，这是 408 追求的最优时间解。

2012 年 408 统考真题 - 算法题 (42)

第 42 题

假定采用带头结点的单链表保存单词，当两个单词有相同的后缀时，可共享存储空间。
设 `str1` 和 `str2` 分别指向两个单词所在单链表的头结点，设计一个时间尽可能高效的算法，找出两个链表共同后缀的起始位置。

2012 年 408 统考真题 - 算法题 (42)

第 42 题

假定采用带头结点的单链表保存单词，当两个单词有相同的后缀时，可共享存储空间。
设 str1 和 str2 分别指向两个单词所在单链表的头结点，设计一个时间尽可能高效的算法，找出两个链表共同后缀的起始位置。

核心观察

如果两个链表有共同后缀，那么从某个结点开始，它们之后的所有结点都是共享的，呈现“Y”字型结构。

算法设计思想

基本设计思想：长度对齐同步法
由于共同后缀的长度相同，两个链表总长度的差异一定出现在共同后缀之前的部分。

1. **测长**：分别遍历两个链表 `str1` 和 `str2`，得到它们的长度 L_1 和 L_2。
2. **对齐**：计算长度差 $\Delta L = |L_1 - L_2|$。令较长链表的指针先向前移动 ΔL 步，使得两个指针剩余的长度相等。
3. **同步对比**：两个指针同时向后移动。在移动过程中，比较指针所指的结点地址是否相同。
4. **输出**：第一个相同的结点地址即为共同后缀的起始位置。

复杂度分析：

- **时间复杂度**：$O(L_1 + L_2)$。需要遍历两次链表（一次测长，一次找交点）。
- **空间复杂度**：$O(1)$。仅需常数个辅助指针和变量。

```c
typedef struct node {
    char data;
    struct node *next;
} NODE;

NODE* findCommonSuffix(NODE *str1, NODE *str2) {
    int len1 = 0, len2 = 0;
    NODE *p = str1->next, *q = str2->next;
    // 1. 统计两个链表的长度
    while (p) { len1++; p = p->next; }
    while (q) { len2++; q = q->next; }
    p = str1->next; q = str2->next;
    // 2. 较长链表先走，使两个指针同步
    if (len1 > len2) for (int k = 0; k < len1 - len2; k++) p = p->next;
    else             for (int k = 0; k < len2 - len1; k++) q = q->next;
    // 3. 同时移动，寻找第一个相同的结点
    while (p != NULL && p != q) {
        p = p->next;
        q = q->next;
    }
    return p; // 若无共同后缀，则返回NULL
}
```

```c
typedef struct node {
    char data;
    struct node *next;
} NODE;

NODE* findCommonSuffix(NODE *str1, NODE *str2) {
    int len1 = 0, len2 = 0;
    NODE *p = str1->next, *q = str2->next;
    // 1. 统计两个链表的长度
    while (p) { len1++; p = p->next; }
    while (q) { len2++; q = q->next; }
    p = str1->next; q = str2->next;
    // 2. 较长链表先走，使两个指针同步
    if (len1 > len2) for (int k = 0; k < len1 - len2; k++) p = p->next;
    else             for (int k = 0; k < len2 - len1; k++) q = q->next;
    // 3. 同时移动，寻找第一个相同的结点
    while (p != NULL && p != q) {
        p = p->next;
        q = q->next;
    }
    return p; // 若无共同后缀，则返回NULL
}
```

关键点拨

注意比较的是结点的**指针地址**（即 `p == q`），而不是结点中的数据域（`p->data == q->data`）。相同的数据不代表是同一个结点，但相同的地址一定是共享存储空间。

2009 年 408 统考真题 - 算法题 (42)

第 42 题

已知一个带有表头结点的单链表，结点结构为 | data | link |，头指针为 `list`。
在不改变链表的前提下，设计算法查找链表中倒数第 k 个位置上的结点（k 为正整数）。
要求：

- 查找成功：输出该结点的 `data` 值，返回 1。
- 查找失败：只返回 0。

核心约束

1. 不允许改变链表结构。　2. 尽可能高效（一次遍历）。

算法设计思想与步骤

基本设计思想：双指针法（前后指针）
通过两个指针之间的“距离差”来定位倒数位置。

1. **初始化**：定义两个指针 p 和 q，初始时均指向头结点的下一个结点（首元结点）。
2. **先行**：指针 p 先向后移动 k 个结点。
   - 若在移动 k 步之前 p 已变为空，说明 $k >$ 链表长度，查找失败，返回 0。
3. **同步**：之后，指针 p 和 q 同时向后移动，直到 p 指向空（NULL）。
4. **定位**：此时，由于 p 比 q 领先了 k 个位置，当 p 到达尾部之后时，q 正好指向倒数第 k 个结点。

时间复杂度：$O(n)$。　**空间复杂度**：$O(1)$。

```c
typedef struct node {
    int data;
    struct node *link;
} NODE;

int findKthFromEnd(NODE *list, int k) {
    NODE *p = list->link, *q = list->link;
    int count = 0;
    // 1. 指针 p 先走 k 步
    while (p != NULL && count < k) {
        p = p->link;
        count++;
    }
    // 2. 如果 count < k，说明链表长度不足 k
    if (count < k) return 0;
    // 3. p 和 q 同步移动
    while (p != NULL) {
        p = p->link;
        q = q->link;
    }
    // 4. 此时 q 指向倒数第 k 个结点
    printf("倒数第 %d 个结点的 data 值为: %d\n", k, q->data);
    return 1;
}
```

考场细节

注意题目要求输出 data 域的值。如果 k 的定义是从 1 开始计数，则 p 先移动 k 步后，q 在 p 为 NULL 时恰好在目标位置。

历年真题

1. 线性结构 (11)
   - 顺序表 (5)
   - 链表 (4)
   - 栈 (1)
     - 2026-42
   - 队列 (1)
2. 树 (4)
3. 图 (10)
4. 排序 + 外排序 (5)
5. 查找 + 高级查找 (6)

2026 年 408 统考模拟 - 栈的应用 (42)

第 42 题

栈的基本操作有出栈和入栈。将序列 $1, 2, 3, \ldots, n$ 依次入栈：

1. $n = 9$ 时，可得序列 $\{2, 3, 1, 6, 4, 7, 5, 8, 9\}$ 吗？可得 $\{2, 3, 1, 4, 6, 5, 7, 8, 9\}$ 吗？
2. 若出栈序列 $P_1, P_2, \ldots, P_n$ 非法，则其中项 $P_i, P_j, P_k (i < j < k)$ 的大小关系？
3. 若 $n = 4$，则以 2 开头的出栈序列个数？
4. 若 $n = k - 1$ 时序列总数为 M，当 $n = k$ 时，以 1 开头、以 2 开头的序列数及总数分别是多少？

问题 (1) 与 (2) 解析

(1) 序列合法性判断：

- **序列 1** $\{2,3,1,6,4,7,5,8,9\}$：当 6 出栈后，栈内剩余 $\{4,5\}$（4 在底）。后续出栈顺序必须是 4 紧跟 5。序列中 6 后是 4 之后才是 $7,5$，非法。
- **序列 2** $\{2,3,1,4,6,5,7,8,9\}$：合法。模拟过程：push(1,2), pop(2), push(3), pop(3), pop(1), push(4), pop(4)...

(2) 非法序列特征（312 禁忌）：出栈序列合法的充要条件是：对于任意 $i<j<k$，不能出现 $P_k<P_j<P_i$ 的情况。即：若 $i<j<k$，且大小关系满足 $P_k<P_j<P_i$，则该序列非法。

问题 (3) 与 (4) 解析

(3) $n = 4$ **以 2 开头的序列**：若第一个出栈的是 2，说明此时 1 已经在栈中。

- 1 可以在 2 之后的任意位置出栈。
- 实际上等价于：固定 2 为首位，剩余 $\{1, 3, 4\}$ 的合法排列。
- 情况分析：2 出栈后，1 必须在 3 之前或紧随其后。计算得共 **3** 种：$(2, 1, 3, 4), (2, 3, 1, 4), (2, 3, 4, 1)$。

(4) 通项推导（卡特兰数）：

- **以 1 开头**：1 入即出，剩下 $k - 1$ 个元素的排列，即 M 个。
- **以 2 开头**：1 必须在 2 之后出，由 (3) 推广，其个数也为 M 个。
- **总序列数**：满足卡特兰数 $C_n = \frac{1}{n+1}\binom{2n}{n}$。

公式总结

n 个元素入栈，出栈序列总数 $f(n) = \sum_{i=0}^{n-1} f(i) \times f(n-1-i)$。

历年真题

1. 线性结构 (11)
   - 顺序表 (5)
   - 链表 (4)
   - 栈 (1)
   - 队列 (1)
     - 2019-42
2. 树 (4)
3. 图 (10)
4. 排序 + 外排序 (5)
5. 查找 + 高级查找 (6)

2019 年 408 统考真题 - 算法题 (42)

第 42 题

设计一个队列满足：初始为空；入队时允许增加空间；出队后空间可复用，总空间只增不减；入、出队复杂度均为 $O(1)$。

回答问题：

1. 选择链式还是顺序存储结构？
2. 画出初始状态，给出队空和队满条件。
3. 画出第一个元素入队后的状态。
4. 给出入队和出队的操作过程。

结构选择与状态定义

(1) 结构选择：应选择**链式存储结构**（具体的说，是带头结点的循环单链表）。顺序存储空间固定，不便于“动态增加且只增不减”。

(2) 初始状态与判空判满：

- **指针设置**： `front` 指向队头结点的前驱（头结点），`rear` 指向队尾结点。
- **初始状态**：只有一个头结点，`front` 和 `rear` 均指向它，且头结点的 `next` 指向自身。
- **队空条件**： `front == rear`。
- **队满条件**： `rear->next == front`（此时需动态开辟新结点）。

入队/出队操作过程

(3) 第一个元素入队后状态：`front` 仍指向头结点，`rear` 指向新插入的第一个结点。若之前空间不足，则 `rear` 指向新开辟的结点并将其插入循环链表。

(4) 基本操作过程：

- **入队**：
  1. 若 `rear->next == front`（队满），则在 `rear` 之后插入一个新结点。
  2. `rear = rear->next`。
  3. 将入队元素写入 `rear` 所指结点的 `data` 域。
- **出队**：
  1. 若 `front == rear`（队空），则报错。
  2. `front = front->next`。
  3. 取出 `front` 所指结点的 `data`。

大题分布

1. 线性结构 (11)
2. 树 (4)
   - 哈夫曼树 (2)
   - 树基础：概念、遍历 (2)
3. 图 (10)
4. 排序 + 外排序 (5)
5. 查找 + 高级查找 (6)

历年真题

1. 线性结构 (11)
2. 树 (4)
   - 哈夫曼树 (2)
     - 2020-42
     - 2014-41
   - 树基础：概念、遍历 (2)
3. 图 (10)
4. 排序 + 外排序 (5)
5. 查找 + 高级查找 (6)

2020 年 408 统考真题 - 树的应用 (42)

第 42 题

任一个字符的编码都不是其它字符编码的前缀，则称这种编码具有前缀特性。现有一组不等长二进制编码，最长为 L 位。

回答问题：

1. 哪种数据结构适宜保存上述具有前缀特性的不等长编码？
2. 简述从 $0/1$ 串到字符串的译码过程。
3. 简述判定某字符集的不等长编码是否具有前缀特性的过程。

2020 年 408 统考真题 - 树的应用 (42)

第 42 题

任一个字符的编码都不是其它字符编码的前缀，则称这种编码具有前缀特性。现有一组不等长二进制编码，最长为 L 位。

回答问题：

1. 哪种数据结构适宜保存上述具有前缀特性的不等长编码？
2. 简述从 0/1 串到字符串的译码过程。
3. 简述判定某字符集的不等长编码是否具有前缀特性的过程。

核心考点

前缀编码在二叉树中的体现：所有叶子结点代表一个字符。

结构选择与译码过程

(1) 数据结构选择：
最适宜的数据结构是**二叉树**。在该二叉树中，左分支代表 0，右分支代表 1。每个编码对应的字符均存储在**叶子结点**中。

(2) 译码过程：

1. **初始化**：指针 p 指向二叉树的根结点。
2. **扫描与移动**：依次读入二进制序列中的位。若读到 0，指针 p 移向左孩子；若读到 1，指针 p 移向右孩子。
3. **输出**：若指针 p 到达叶子结点，输出该结点存储的字符，并将指针 p 重新指向根结点，开始下一次译码。
4. **循环**：重复上述过程，直至二进制序列扫描完毕。

前缀特性判定过程

(3) 判定过程：
通过构造或遍历对应的二叉树来判定。将所有字符的编码按 0/1 规则依次插入二叉树中：

- **规则**：每一个字符的编码路径上，除了最后一个比特位对应叶子结点外，路径上的其他中间位不能是已经存在的叶子结点。
- **冲突判定**：在插入过程中，若出现以下两种情况之一，则不具有前缀特性：

  情况 A：某个字符编码的路径上遇到了一个已存在的叶子结点（说明它是其他编码的前缀）。
  情况 B：某个新编码插入完成后，其对应的结点不是叶子结点（说明其他已有编码是它的前缀）。
- **结论**：若所有编码都能成功插入且均为叶子结点，则该编码具有前缀特性。

总结

前缀特性的本质是：在二叉树中，任何一个字符对应的结点都不能是另一个字符对应结点的祖先。

2014 年 408 统考真题 - 算法题 (41)

第 41 题

二叉树的带权路径长度（WPL）是所有叶结点的带权路径长度之和。给定一棵采用二叉链表存储的二叉树 T。
叶结点的 `weight` 域保存该结点的非负权值。设计求 T 的 WPL 的算法。

结点结构：

left	weight	right

核心公式

$WPL = \sum_{i\in} weight_i \times depth_i$
其中根结点的深度通常定义为 0（或 1，需在算法中保持一致）。

基本设计思想：递归遍历
采用先序遍历或后序遍历的递归框架：

1. **递归参数**：传入当前结点指针 `root` 和当前深度 `depth`。
2. **终止条件**：若当前结点为空，返回 0。
3. **叶子判定**：若当前结点为叶子结点（左右孩子均为空），则返回该结点的 `weight` 乘以 `depth`。
4. **递归步**：若不是叶子结点，则返回左子树的 WPL 与右子树的 WPL 之和。

复杂度：时间复杂度 $O(n)$，空间复杂度 $O(h)$（h 为树高）。

```
1 typedef struct BiTNode {
2     int weight;
3     struct BiTNode *left, *right;
4 } BiTNode, *BiTree;
```

```c
// 计算 WPL 的递归辅助函数
int wpl_Recursive(BiTree root, int depth) {
    if (root == NULL) return 0; // 空结点返回 0
    // 若为叶子结点，返回其带权路径长度
    if (root->left == NULL && root->right == NULL) {
        return root->weight * depth;
    }
    // 若为分支结点，递归计算左右子树之和
    return wpl_Recursive(root->left, depth + 1) +
           wpl_Recursive(root->right, depth + 1);
}

int WPL(BiTree root) {
    return wpl_Recursive(root, 0); // 根结点深度设为 0
}
```

备考提示

1. **深度定义**：若题目未指明，根结点深度设为 0 或 1 均可，但需在逻辑中自洽。
2. **非递归解法**：亦可采用层序遍历，利用队列记录每个结点的深度，适合在需要避免深递归导致的栈溢出时使用。

历年真题

1. 线性结构 (11)
2. 树 (4)
   - 哈夫曼树 (2)
   - 树基础：概念、遍历 (2)
     - 2017-41
     - 2016-42
3. 图 (10)
4. 排序 + 外排序 (5)
5. 查找 + 高级查找 (6)

2017 年 408 统考真题 - 算法题 (41)

第 41 题

设计一个算法，将给定的表达式树（二叉树）转换为等价的中缀表达式并输出。

- 树 1 输出：$(a+b)*(c*(-d))$
- 树 2 输出：$(a*b)+(-(c-d))$

```
1 typedef struct node {
2     char data[10];
3     struct node *left, *right;
4 } BTree;
```

核心考点

基于中序遍历的改进：通过判断“非叶子结点”来控制括号的打印。

算法设计思想

基本设计思想：递归中序遍历
表达式树的中序遍历序列即为中缀表达式。括号的添加规律如下：

1. **根结点与叶子结点**：不需要添加括号。
2. **其他中间结点（操作符）**：在遍历其左子树之前打印左括号 '('，遍历完其右子树之后打印右括号 ')'。

具体实现步骤：

- 定义递归函数 BtreeToInfix(root，depth)，根结点深度设为 1。
- 若结点为叶子，直接输出 data。
- 若为非叶子：
  1. 若当前不是根（depth > 1），输出 '('。
  2. 递归遍历左子树 → 输出 data → 递归遍历右子树。
  3. 若当前不是根（depth > 1），输出 ')'。

```c
void BtreeToE(BTree *root, int depth) {
    if (root == NULL) return;
    // 情况1：叶子结点
    if (root->left == NULL && root->right == NULL) {
        printf("%s", root->data);
    }else { // 情况2：非叶子结点（操作符）
        if (depth > 1) printf("("); // 非根结点的操作符左侧加括号
        BtreeToE(root->left, depth + 1); // 遍历左子树
        printf("%s", root->data);        // 输出操作符
        BtreeToE(root->right, depth + 1);// 遍历右子树
        if (depth > 1) printf(")"); // 非根结点的操作符右侧加括号
    }
}
// 主程序调用入口
void BtreeToInfix(BTree *root) {
    BtreeToE(root, 1);
}
```

逻辑总结

这种方法生成的括号是“冗余”但“等价”的，能通过括号反映所有运算次序。关键在于判断 `depth > 1` 和 **非叶子结点**。

2016 年 408 统考真题 - 树的性质 (42)

第 42 题

若一棵非空 k ($k \geq 2$) 叉树 T 中每个非叶结点都有 k 个孩子，则称 T 为正则 k 叉树。请回答下列问题并给出推导过程：

1. 若 T 有 m 个非叶结点，则 T 中的叶结点有多少个？
2. 若 T 的高度为 h ($h = 1$ 为单结点)，则 T 的结点数最多为多少个？最少为多少个？

2016 年 408 统考真题 - 树的性质 (42)

第 42 题

若一棵非空 k $(k \geq 2)$ 叉树 T 中每个非叶结点都有 k 个孩子，则称 T 为正则 k 叉树。请回答下列问题并给出推导过程：

1. 若 T 有 m 个非叶结点，则 T 中的叶结点有多少个？
2. 若 T 的高度为 h $(h = 1$ 为单结点$)$，则 T 的结点数最多为多少个？最少为多少个？

核心工具

1. 树的总结点数 $n = n_0 + n_k$
2. 树的分支数与总结点数关系：$n =$ $+1$

问题 (1) 推导：叶结点数计算

推导过程：

1. **按分支数计算**：每个非叶结点都有 k 个孩子，故总的分支数为 $m \times k$。
2. **按结点总数计算**：树的总结点数 $n = \quad + 1 = m \times k + 1$。
3. **按度数分类**：正则 k 叉树中只有度为 0（叶子）和度为 k（非叶子）的结点。设叶子结点数为 n_0，则有：
$$n = n_0 + m$$
4. **联立方程**：
$$n_0 + m = m \times k + 1$$
$$n_0 = m(k-1) + 1$$

结论：叶结点数为 $m(k-1)+1$。

问题 (2) 推导：结点数极值

- **最多结点数（满 k 叉树）**：当树为高度为 h 的满 k 叉树时，结点数最多。第 i 层结点数为 k^{i-1}。

$$n_{max} = 1 + k + k^2 + \cdots + k^{h-1} = \frac{k^h - 1}{k - 1}$$

- **最少结点数（瘦高型）**：正则 k 叉树要求非叶结点必须有 k 个孩子。为了使总数最少，每一层应只有一个非叶结点。
  1. 第 1 层（根）：1 个非叶结点。
  2. 第 2 层到第 $h-1$ 层：每层只有 1 个非叶结点，其余 $k-1$ 个均为叶子。
  3. 第 h 层：全部为叶子结点（共 k 个）。
  4. **总结点数**：前 $h-1$ 层共有 $h-1$ 个非叶结点，最后一层有 k 个叶子。

$$n_{min} = (h-1) \times k + 1$$

注意：最少结点情况下的推导：除根外，每增加一层，至少要增加 k 个结点（其中一个是下一层的父结点）。

大题分布

1. 线性结构 (11)
2. 树 (4)
3. 图 (10)
   - 邻接矩阵本身 (3)
   - 最短路径 (2)
   - 最小生成树 (2)
   - 关键路径 (2)
   - 拓扑排序 (1)
4. 排序 + 外排序 (5)
5. 查找 + 高级查找 (6)

历年真题

1. 线性结构 (11)
2. 树 (4)
3. 图 (10)
   - 邻接矩阵本身 (3)
     - 2023-41
     - 2021-41
     - 2015-42
   - 最短路径 (2)
   - 最小生成树 (2)
   - 关键路径 (2)
   - 拓扑排序 (1)
4. 排序 + 外排序 (5)

2023 年 408 统考真题 - 算法题 (41)

第 41 题

已知有向图 G 采用邻接矩阵存储。将图中出度大于入度的顶点称为 **K 顶点**。
设计算法 `int printVertices(MGraph G)` 输出所有 K 顶点并返回其个数。
图示例：顶点 a $(out = 2, in = 1)$ 和 b $(out = 2, in = 1)$ 是 K 顶点。

邻接矩阵性质

- 第 i 行非零元素的个数 $=$ 顶点 v_i 的**出度**。
- 第 i 列非零元素的个数 $=$ 顶点 v_i 的**入度**。

算法设计思想

通过两次嵌套循环遍历邻接矩阵，分别统计每个顶点的入度和出度。

1. **初始化**：定义两个数组 `inDegree` 和 `outDegree`，长度均为 `numVertices`，初始化为 0。
2. **遍历矩阵**：使用双重循环遍历 `Edge[i][j]`：
   - 若 `Edge[i][j] == 1`，则 `outDegree[i]++`，同时 `inDegree[j]++`。
3. **判断与输出**：遍历顶点表，若满足 `outDegree[i] > inDegree[i]`：
   - 输出该顶点 `VerticesList[i]`；
   - 计数器 `count` 加 1。
4. **返回**：返回 `count`。

复杂度：时间复杂度 $O(n^2)$，空间复杂度 $O(n)$，其中 n 为顶点数。

```c
int printVertices(MGraph G) {
    int n = G.numVertices;
    int inDegree[MAXV] = {0};  // 存储各顶点入度
    int outDegree[MAXV] = {0}; // 存储各顶点出度
    int count = 0;
    // 1. 扫描邻接矩阵，计算度数
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (G.Edge[i][j] != 0) { // 存在从 i 到 j 的有向边
                outDegree[i]++;      // i 的出度增加
                inDegree[j]++;       // j 的入度增加
            }
    // 2. 统计并输出 K 顶点
    printf("K 顶点为: ");
    for (int i = 0; i < n; i++)
        if (outDegree[i] > inDegree[i]) {
            printf("%c ", G.VerticesList[i]);
            count++;
        }
    printf("\n");
    return count;
}
```

优化提示

如果为了追求极致的空间复杂度（$O(1)$ 辅助空间），可以对每个顶点 i 单独扫描其所在的行和列，但这会将时间复杂度内的常数项增大（每列被重复扫描）。

2021 年 408 统考真题 - 算法题 (41)

第 41 题

已知无向连通图 G 采用邻接矩阵存储。若 G 中度为奇数的顶点个数为不大于 2 的偶数（即 0 或 2），则称 G 存在包含所有边且长度为 $|E|$ 的路径（EL 路径）。
设计算法 `int IsExistEL(MGraph G)`，判断 G 是否存在 EL 路径。

判定准则

存在 EL 路径 $\Longleftrightarrow$ 度为奇数的顶点个数 $\in \{0, 2\}$。
在无向图邻接矩阵中，顶点 v_i 的度 $=$ 第 i 行（或列）非零元素的个数。

算法设计思想

基本设计思想：
根据题目给出的 EL 路径存在条件，算法的核心是统计无向图中每个顶点的度，并计算度为奇数的顶点总数。

1. **初始化**：定义计数器 `oddCount = 0`。
2. **计算度数**：依次扫描邻接矩阵的每一行：
   - 对于第 i 行，遍历该行所有元素，累加非零元素的个数，得到顶点 v_i 的度 `degree`。
3. **奇数度判定**：检查 `degree` 是否为奇数（`degree % 2 != 0`），若是，则 `oddCount++`。
4. **最终判断**：扫描完所有顶点后，若 `oddCount` 等于 0 或 2，则返回 1；否则返回 0。

复杂度分析：

- **时间复杂度**：$O(n^2)$。需要遍历整个 $n \times n$ 的邻接矩阵。
- **空间复杂度**：$O(1)$。仅需常数个辅助变量（若不存储所有度数）。

```
int IsExistEL(MGraph G) {
    int oddCount = 0; // 记录度为奇数的顶点个数
    // 遍历邻接矩阵，计算每个顶点的度
    for (int i = 0; i < G.numVertices; i++) {
        int degree = 0;
        for (int j = 0; j < G.numVertices; j++)
            if (G.Edge[i][j] != 0)
                degree++; // 计算第 i 个顶点的度
        // 判断当前顶点的度是否为奇数
        if (degree % 2 != 0) oddCount++;
    }
    // 根据 EL 路径存在条件进行判断
    if (oddCount == 0 || oddCount == 2) {
        return 1;
    } else {
        return 0;
    }
}
```

考场细节

注意题目已明确图 G 是“连通图”，因此我们不需要再手动使用 DFS 或 BFS 检查连通性，只需专注于“奇数度顶点”的统计即可。

2015 年 408 统考真题 - 图的性质 (42)

第 42 题

已知含有 5 个顶点的无向图 G（顶点为 0, 1, 2, 3, 4）。

1. 写出图 G 的邻接矩阵 A。
2. 求 A^2，并说明 A^2 中位于 0 行 3 列元素值的含义。
3. 若图有 n 个顶点，邻接矩阵为 B，则 $B^m(2 \leq m \leq n)$ 中非零元素的含义是什么？

核心定理

设 G 的邻接矩阵为 A，则 A^m 中的元素 $a_{ij}^{(m)}$ 表示从顶点 i 到顶点 j 长度为 m 的**路径条数**。

(1) 邻接矩阵 A：根据图的连通情况 $(0-1, 0-2, 0-3, 1-2, 2-3, 3-4)$，矩阵如下：

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

(2) 矩阵 A^2 计算：利用矩阵乘法 $A^2 = A \times A$，计算得：

$$A^2 = \begin{pmatrix} 3 & 1 & 2 & 1 & 1 \\ 1 & 2 & 1 & 2 & 0 \\ 2 & 1 & 3 & 1 & 1 \\ 1 & 2 & 1 & 3 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

0 行 3 列元素值 (1) 的含义：
表示从顶点 0 到顶点 3 长度为 2 的路径共有 1 条（该路径为 $0 \to 2 \to 3$）。

问题 (3) 推导：矩阵幂的通用含义

(3) B^m 中非零元素的含义：
若邻接矩阵 B 的 m 次幂 B^m 中位于 i 行 j 列的元素 $b_{ij}^{(m)} \neq 0$，其含义为：

- **路径存在性**：表示从顶点 i 到顶点 j 之间存在长度为 m 的路径。
- **路径计数**：该元素的具体数值代表了从顶点 i 到顶点 j 长度恰好为 m 的路径的总条数。

名师点拨

注意：路径中允许包含重复的顶点和边。例如在 A^2 中，对角线元素 $a_{ii}^{(2)}$ 表示从 i 出发走 2 步回到 i 的路径数，实质上等于顶点 i 的度数。

历年真题

1. 线性结构 (11)
2. 树 (4)
3. 图 (10)
   - 邻接矩阵本身 (3)
   - 最短路径 (2)
     - 2014-42
     - 2009-41
   - 最小生成树 (2)
   - 关键路径 (2)
   - 拓扑排序 (1)
4. 排序 + 外排序 (5)

2014 年 408 统考真题 - 综合应用题 (42)

第 42 题

题 42 表展示了路由器 R1 维护的链路状态信息（LSI），题 42 图是基于此构造的网络拓扑。

1. 本题中的网络可抽象为数据结构中的哪种结构？
2. 设计合理的链式存储结构保存 LSI，并给出数据定义和示意图。
3. 按照 Dijkstra 算法策略，给出 R1 到达各子网的最短路径及费用。

(1) 结构抽象

本题中的网络（路由器作为顶点，链路作为边）可以抽象为数据结构中的**带权图（无向图）**。

为了保存 LSI，我们需要一个“顶点表”存储路由器信息，每个顶点关联一个“链路表”存储邻居信息，以及一个“网路表”存储直连网段信息。

```
typedef struct LinkNode {
    int ID;              // 所连路由器 ID
    int IP;              // 本地接口 IP
    int metric;          // 费用
    struct LinkNode *next;
} LinkNode;  // 链路信息/邻接边

typedef struct NetNode {
    char prefix[20];     // 网络前缀
    int metric;          // 费用
    struct NetNode *next;
} NetNode;  // 网络前缀信息

typedef struct RouterNode {
    int RouterID;        // 路由器 ID
    LinkNode *lnext;     // 指向链路表
    NetNode *nnext;      // 指向网路表
    struct RouterNode *next;
} RouterNode; // 路由器结点/顶点
```

存储示意图： R1 → {L: R2, R3} → {N: 192.168.1.0}

(3) Dijkstra 最短路径求解

R1 (10.1.1.1) 到各子网的最短路径分析：

1. **到 192.168.1.0/24**： R1 直连。路径：R1 → 子网。费用：**1**。
2. **到 192.168.2.0/24 (由 R2 通告)**：路径：R1 → R2 → 子网。费用：$3(R1 - R2) + 1 = \mathbf{4}$。
3. **到 192.168.3.0/24 (由 R3 通告)**：路径：R1 → R3 → 子网。费用：$2(R1 - R3) + 1 = \mathbf{3}$。
4. **到 192.168.4.0/24 (由 R4 通告)**：
   - 经 R2：$R1 \to R2 \to R4 \to$ ，费用 $3 + 4 + 1 = 8$。
   - 经 R3：$R1 \to R3 \to R4 \to$ ，费用 $2 + 6 + 1 = 9$。

   最短路径：R1 → R2 → R4 → 子网。费用：**8**。

解题关键

OSPF 链路状态算法计算的是从源路由器到目的网络（子网）的端到端费用。总费用 = 沿途所有出接口的 Metric 之和 + 目标网络的 Metric。

2009 年 408 统考真题 - 算法辨析 (41)

第 41 题

带权图（权值非负）的最短路径问题。现有一种算法：

① 最短路径初始仅包含初始顶点，令当前顶点 u 为初始顶点；

② 选择离 u **最近**且尚未在路径中的一个顶点 v，加入路径，修改当前顶点 $u = v$；

③ 重复步骤②，直到 u 是目标顶点。

请问：上述方法能否求得最短路径？说明理由。

直觉判断

该方法属于典型的**贪心算法**（局部最优），而非求解全局最短路径的经典算法。

结论与反例说明

结论：该方法**不能**求得最短路径。
原因：该算法每一步只考虑从“当前顶点”出发的最近邻居，这是一种局部贪心策略，忽略了已经加入路径的其他顶点可能存在的更短分支。

反例举证：考虑起始点 S 到目标点 D 的路径。

- 顶点与边：$V = \{S, A, D\}$
- 边权：$(S, A) = 1, (A, D) = 10, (S, D) = 5$
- **该算法执行**：
  1. 从 S 出发，最近的邻居是 A（权值 1），路径变为 $S \to A$。
  2. 从 A 出发，最近且唯一的邻居是 D（权值 10），路径变为 $S \to A \to D$。
  3. **总路径长度**：**11**。
- **实际最短路径**：直接走 $S \to D$，**总长度为 5**。

原理深度剖析

为什么该算法会失效？

- **题目方法**：类似“走迷宫”只看脚下，一旦选定一个顶点，就从该新顶点继续“探路”，不再回头。
- **Dijkstra**：每次从 **已确定的最短路径集合**作为一个整体向外探测，选择的是离 **源点**最近的顶点。

比较维度	题目所述方法	Dijkstra 算法
核心逻辑	局部贪心（从当前点出发）	全局贪心（从源点出发）
回溯机制	无（当前点固定前移）	有（动态更新其他点的距离）
最优性保证	不保证	保证（权值非负时）

考场启示

在图论中，“最近”的参照物极其重要。Dijkstra 的精髓在于 $dist[v] = min(dist[v], dist[u] + w(u,v))$ 的松弛操作，这是本题算法所缺失的。

历年真题

1. 线性结构 (11)
2. 树 (4)
3. 图 (10)
   - 邻接矩阵本身 (3)
   - 最短路径 (2)
   - 最小生成树 (2)
     - 2018-42
     - 2017-42
   - 关键路径 (2)
   - 拓扑排序 (1)
4. 排序 + 外排序 (5)

2018 年 408 统考真题 - 综合应用题 (42)

第 42 题

拟连通 BJ、CS、XA、QD、JN、NJ、TL、WH 等 8 个城市，边权表示光缆铺设费用。

1. 给出所有可能的最经济铺设方案（最小生成树），并计算总费用。
2. 给出图的存储结构及求解算法名称。
3. H1(TL) 向 H2(BJ) 发送 TTL=5 的分组，H2 是否能收到？

核心考点

最经济方案 $\Longleftrightarrow$ 最小生成树 (MST)。注意图中是否存在等权边导致的多种方案。

问题 (1) 与 (2) 解析

(1) 最经济方案：使用 Prim 或 Kruskal 算法。图中总共 8 个顶点，MST 应包含 7 条边。经过计算，满足条件的最小生成树方案有两类（因某些权值为 2 的边可选）：

- **边集合**： (XA-BJ, 2), (XA-TL, 2), (BJ-JN, 3), (JN-QD, 2), (JN-NJ, 3), (NJ-WH, 3), (WH-CS, 4)。
- **另一种可能**：若 (NJ-CS, 4) 替换 (WH-CS, 4)。

总费用： $2+2+3+2+3+3+4=\mathbf{19}$。
(2) 存储结构与算法：

- **存储结构**：邻接矩阵或邻接表。
- **算法名称**： Prim 算法或 Kruskal 算法。

问题 (3) 网络传输分析

TTL 原理：IP 分组每经过一个路由器，TTL 减 1。若 TTL 减为 0 且未到达目的地，则分组被丢弃。
路径分析（基于最经济方案）：在 (1) 得到的生成树中，从 TL 到 BJ 的路径为：**TL → XA → BJ**

1. H1 发出分组，进入 TL 路由器。
2. TL 转发至 XA 路由器：TTL $= 5 - 1 = 4$。
3. XA 转发至 BJ 路由器：TTL $= 4 - 1 = 3$。
4. BJ 路由器直接交付给 H2。

结论：由于路径仅需经过 2 跳路由器转发，TTL 剩余 3，因此 **H2 可以收到该 IP 分组**。

得分关键

1. 正确画出 MST 是基础。2. TTL 减 1 的操作发生在路由器**转发**时。3. 即使在最不利的 MST 结构中，TL 到 BJ 的距离也未超过 5 跳。

2017 年 408 统考真题 - 图论应用 (42)

第 42 题

使用 Prim 算法求带权连通图的最小生成树（MST）：

1. 对下图 G，从顶点 A 开始，依次给出算法选出的边。
2. 图 G 的 MST 是唯一的吗？
3. 对任意带权连通图，满足什么条件时，其 MST 是唯一的？

核心思想：Prim 算法

从初始顶点开始，每次选择连接“已选顶点集”与“未选顶点集”之间权值最小的边。

问题 (1) 解析：Prim 执行过程

从顶点 **A** 开始构建：

1. **初始**：集合 $U = \{A\}$。**可选边** $(A, E) : 5, (A, B) : 6$。**选 (A, E)**。
2. **当前**：$U = \{A, E\}$。**可选边** $(E, B) : 4, (E, D) : 4, (E, C) : 6, (A, B) : 6$。
   选 (E, B) 或 (E, D)。假设选 **(E, B)**。
3. **当前**：$U = \{A, E, B\}$。**可选边** $(B, C) : 4, (E, D) : 4, (E, C) : 6, (A, B) : 6$。
   选 (E, D)。
4. **当前**：$U = \{A, E, B, D\}$。**可选边** $(B, C) : 4, (D, C) : 5, (E, C) : 6$。
   选 (B, C)。

最终选出的边依次为：
(A, E), (E, B), (E, D), (B, C)　(或 (A, E), (E, D), (E, B), (B, C))

问题 (2) 与 (3) 解析：唯一性讨论

(2) 图 G 的 MST 是否唯一？
不唯一。
在构造过程的第 2 步，存在两条权值相等的最小边 (E, B) 和 (E, D) 可供选择，且它们不属于同一回路。虽然最终 MST 的总权值相同（$5+4+4+4=17$），但所包含的边集可能不同（例如 (E, B) 和 (E, D) 的选择次序不影响结果，但若存在其他等权竞争则可能产生不同形态）。
(3) MST 唯一的充分条件：
对任意带权连通图，若图中**各边的权值均不相等**，则其最小生成树是唯一的。

深度思考

注意：“各边权值互不相等”是唯一性的**充分条件**而非必要条件。即使有权值相等的边，该图的 MST 也有可能是唯一的（只要等权边不参与 MST 的竞争）。

历年真题

1. 线性结构 (11)
2. 树 (4)
3. 图 (10)
   - 邻接矩阵本身 (3)
   - 最短路径 (2)
   - 最小生成树 (2)
   - 关键路径 (2)
     - 2025-42
     - 2011-41
   - 拓扑排序 (1)
4. 排序 + 外排序 (5)

2025 年 408 统考模拟 - AOE 网分析 (42)

第 42 题

已知一 AOE 网描述了 12 个工程活动。各活动及其持续时间如下：
活动： a=2, b=5, c=1, d=3, e=3, f=4, g=1, h=1, j=1, k=2, m=4, n=3。

1. 完成工程的最短时间是多少？哪些是关键活动？
2. 与活动 e 同时进行的活动可能有哪些？
3. 时间余量最大的活动及其余量？
4. 若 b 延迟到时刻 6 开始，如何调整以保证不延期？

核心概念

最短工期 = 关键路径长度。关键活动 = 最早开始时间 (ee) 等于最迟开始时间 (el) 的活动。

问题 (1) 与 (2) 解析

(1) 最短时间与关键活动：通过计算事件的最早发生时间 Ve 和最迟发生时间 Vl：

- **关键路径**：$1 \to 3 \to 4 \to 6 \to 7$ (活动路径：$b \to f \to j \to n$)。
- **工期**：$5(b) + 4(f) + 1(j) + 3(n) = \mathbf{13}$。
- **关键活动**：b, f, j, n。

(2) 与活动 e 同时进行的活动：活动 e 位于 $2 \to 4$。计算其时间区间：

- e 的最早开始时间：$ee(e) = Ve(2) = 2$；最早完成时间：5。
- e 的最迟完成时间：$Vl(4) = 9$；最迟开始时间：6。
- **可能同时进行的活动**：在区间 $[2, 9]$ 内进行的非冲突活动，如 d, f, g, h 等（需根据拓扑偏序判断）。

(3) 最大时间余量：时间余量 $Slack(i) = el(i) - ee(i)$。

- 计算各非关键活动余量，发现活动 c 或 g 等路径较短处的活动余量较大。
- 经计算，$Slack(a) = (Vl(2) - weight(a)) - Ve(1) = (6-2) - 0 = 4$。
- **最大余量活动**：活动 a，余量为 **4**（具体数值随拓扑图权重分布而定）。

(4) 进度动态调整：

- **b 的持续时间限制**： b 在时刻 6 开始，后续路径为 $f \to j \to n$（总长 $4+1+3=8$）。要保证 13 时刻完工，则 $6 + weight(b') + 8 \leq 13$。得出 $weight(b') \leq -1$（逻辑冲突），这意味着**若不压缩后续活动，b 无论如何调整都会延期**。
- **压缩策略**：若保持 $b = 5$，延迟到 6 开始，则 $6+5+f+j+n = 13$。需压缩关键路径上的后续活动：将 f, j 或 n 的持续时间总计压缩 **3** 个单位。

考场技巧

如果一个关键活动延迟开始，它会直接推迟整个工程。此时只有压缩该活动本身或其后续的所有关键路径分支，才能挽回工期。

2011 年 408 统考真题 - 综合应用题 (41)

第 41 题

已知有 6 个顶点的有向带权图 G，其邻接矩阵 A 为上三角矩阵（不含对角线），按行优先保存在一维数组中：[4，6，∞，∞，∞，5，∞，∞，∞，4，3，∞，∞，3，3]

1. 写出图 G 的邻接矩阵 A。
2. 画出有向带权图 G。
3. 求图 G 的关键路径，并计算长度。

(1) 邻接矩阵 A 的还原

$$A = \begin{pmatrix} 0 & 4 & 6 & \infty & \infty & \infty \\ 0 & 0 & 5 & \infty & \infty & \infty \\ 0 & 0 & 0 & 4 & 3 & \infty \\ 0 & 0 & 0 & 0 & \infty & 3 \\ 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(2) 有向带权图 G 的绘制

根据矩阵 A 中的非 ∞ 元素绘图：

- $0 \to 1(4), 0 \to 2(6)$
- $1 \to 2(5)$
- $2 \to 3(4), 2 \to 4(3)$
- $3 \to 5(3)$
- $4 \to 5(3)$

图的特征

这是一个典型的 AOE 网。源点为 0，汇点为 5。

(3) 关键路径与长度：计算事件（顶点）的最早发生时间 *Ve* 和最迟发生时间 *Vl*：

- **Ve(0)=0**
- **Ve(1)=4**
- **Ve(2)=max{4+5, 0+6} = 9**
- **Ve(3)=9+4 = 13**, **Ve(4)=9+3 = 12**
- **Ve(5)=max{13+3, 12+3} = 16**

反向计算 Vl： $Vl(5) = 16, Vl(3) = 13, Vl(4) = 13, Vl(2) = 9, Vl(1) = 4, Vl(0) = 0$。
确定关键活动： $ee = el$ 的活动。

- 路径 1: $0 \to 1 \to 2 \to 3 \to 5$ (长度: $4 + 5 + 4 + 3 = 16$)
- 路径 2: $0 \to 1 \to 2 \to 4 \to 5$ (长度: $4 + 5 + 3 + 3 = 15$)

结论

关键路径： (0, 1), (1, 2), (2, 3), (3, 5) 或写为 **0 → 1 → 2 → 3 → 5**
关键路径长度： **16**

历年真题

1. 线性结构 (11)
2. 树 (4)
3. 图 (10)
   - 邻接矩阵本身 (3)
   - 最短路径 (2)
   - 最小生成树 (2)
   - 关键路径 (2)
   - 拓扑排序 (1)
     - 2024-41
4. 排序 + 外排序 (5)

2024 年 408 统考真题 - 算法题 (41)

第 41 题

神州十七号等复杂工程可抽象为有向图的拓扑序列问题。已知有向图 G 采用邻接矩阵存储。设计算法 `int uniquely(MGraph G)`：判定 G 是否存在唯一的拓扑序列，若是则返回 1，否则返回 0。

核心结论

有向图存在唯一拓扑序列的充要条件是：在拓扑排序的每一步中，入度为 0 的顶点有且仅有一个。

算法设计思想

基本设计思想：改进的拓扑排序算法

1. **初始化**：统计所有顶点的入度，存入数组 `indegree`。将所有入度为 0 的顶点入栈（或队列）。
2. **判唯一性（起始）**：如果初始时栈中顶点个数不为 1，则拓扑序列不唯一或不存在，返回 0。
3. **循环处理**：
   - 当栈不空时，弹出顶点 v，并记录已访问顶点数 `count`。
   - 遍历 v 的所有邻接点，将它们的入度减 1。
   - 若某个邻接点的入度减为 0，则入栈。
   - **关键判定**：在每轮入栈操作结束后，检查栈内元素个数。若某时刻栈中入度为 0 的顶点数大于 1，则说明存在多个并行的活动，拓扑序列不唯一，返回 0。
4. **最终判定**：若成功遍历所有顶点（`count == G.numVertices`），说明存在唯一序列，返回 1。

```
int uniquely(MGraph G) {
    int indegree[MAXV] = {0};
    int stack[MAXV], top = -1;
    int count = 0;
    // 1. 计算所有顶点的入度（邻接矩阵按列累加）
    for (int j = 0; j < G.numVertices; j++)
        for (int i = 0; i < G.numVertices; i++)
            if (G.Edge[i][j] != 0) indegree[j]++;
    // 2. 将所有入度为 0 的顶点入栈
    for (int i = 0; i < G.numVertices; i++)
        if (indegree[i] == 0) stack[++top] = i;
    // 3. 拓扑排序主循环
    while (top != -1) {
        if (top > 0) return 0; // 若栈中元素多于1个，序列不唯一
        int v = stack[top--];
        count++;
        for (int i = 0; i < G.numVertices; i++)
            if (G.Edge[v][i] != 0)
                if (--indegree[i] == 0)
                    stack[++top] = i;
    }
    if (count == G.numVertices) return 1; // 存在且唯一
    else return 0; // 存在回路，序列不完整
}
```

复杂度分析

时间复杂度：$O(V^2)$。邻接矩阵计算入度需 $O(V^2)$，拓扑排序过程涉及双重循环（或遍历矩阵行）同样为 $O(V^2)$。
空间复杂度：$O(V)$。需辅助数组存储入度和栈。

大题分布

1. 线性结构 (11)
2. 树 (4)
3. 图 (10)
4. **排序 + 外排序 (5)**
   - **排序 (3)**
   - **外排序 (2)**
5. 查找 + 高级查找 (6)

历年真题

1. 线性结构 (11)
2. 树 (4)
3. 图 (10)
4. 排序 + 外排序 (5)
   - 排序 (3)
     - 2022-42
     - 2021-42
     - 2016-43
   - 外排序 (2)
5. 查找 + 高级查找 (6)

2022 年 408 统考真题 - 排序与查找 (42)

第 42 题

现有 n（$n > 100,000$）个数保存在一维数组 M 中，需要查找 M 中最小的 10 个数。

1. 设计一个完成上述查找任务的算法，要求平均情况下的比较次数尽可能少。简述算法思想。
2. 说明你所设计的算法平均情况下的时间复杂度和空间复杂度。

核心考点：Top-K 问题

当 $n \gg k$ 时，不宜进行全排序。应使用局部选优的数据结构：**堆 (Heap)**。

算法设计思想：大根堆过滤法

查找**最小**的 k 个数，最适宜采用容量为 k 的**大根堆**。

1. **建堆**：取数组 M 的前 10 个数，构造一个最大容量为 10 的大根堆。
2. **遍历**：从第 11 个数开始，逐个与堆顶元素 M_{max} 比较：
   - 若当前元素 $x < M_{max}$，说明 x 有可能属于最小的 10 个数。
   - **更新**：用 x 替换堆顶元素，并执行一次向下调整（Sift Down）以维持大根堆性质。
   - 若 $x \geq M_{max}$，则直接丢弃，继续看下一个。
3. **结束**：遍历完所有 n 个元素后，堆中留下的 10 个数即为所求。

关键点

问：为什么找“最小”用“大”根堆？
答：堆顶是 10 个候选者中最大的（守门员），只有比它小的数才有资格入选。

复杂度分析与评价

设查找最小的 k 个数，总元素个数为 n：

- **时间复杂度**： $O(n \log k)$
  - 建堆时间： $O(k)$。
  - 遍历与调整：共 $n-k$ 次比较，每次调整时间 $O(\log k)$。
  - 本题中 $k=10$ 为常数，故实际效率极高。
- **空间复杂度**： $O(k)$
  - 仅需额外 k 个元素的空间来维护堆，与 n 的大小无关。这对于内存受限的海量数据场景极其关键。

方案对比

若采用快速排序（$O(n \log n)$）或全选排序（$O(nk)$），在 $n=10^5$ 规模下，其比较次数和内存压力远超堆算法。

2021 年 408 统考真题 - 算法分析 (42)

第 42 题

已知排序算法 cmpCountSort，其逻辑为：通过双重循环比较数组 a 中每两个元素，统计比当前元素小的元素个数存入 count 数组，最后按 count 的值将元素放入 b。

1. 若 $a = \{25, -10, 25, 10, 11, 19\}$，求调用后的数组 b。
2. 算法执行过程中，元素间的比较次数是多少？
3. 该算法是稳定的吗？若是，简述理由；否则，给出修改方案。

核心逻辑

该算法通过两两比较确定每个元素在有序序列中的“排名”（即 count 值）。

(非经典) 排序算法 cmpCountSort

```
1  void cmpCountSort(int a[], int b[], int n)
2  {
3      int i, j, *count;
4      count = (int *) malloc(sizeof(int) * n)
5      for (i = 0; i < n; i++) count[i] = 0;
6      for (i = 0; i < n - 1; i++)
7          for (j = i + 1; j < n; j++)
8              if (a[i] < a[j]) count[j]++;
9              else count[i]++;
10     for (i = 0; i < n; i++) b[count[i]]= a[i];
11     free(count);
12 }
```

问题 (1) 与 (2) 解析

(1) 数组 b 的内容计算：统计 $a = \{25_1, -10, 25_2, 10, 11, 19\}$ 的 count 值：

- $a[0] = 25_1$：比它小的有四个，与 25_2 比较时执行 `count[0]++`，故 `count[0]=5`。
- $a[1] = -10$：最小，`count[1]=0`。
- $a[2] = 25_2$：与 25_1 比较时 `count[2]` 不加，仅加其余四个，故 `count[2]=4`。
- $a[3] = 10$：比它小的只有 -10，故 `count[3]=1`。
- $a[4] = 11$：比它小的有 -10, 10，故 `count[4]=2`。
- $a[5] = 19$：比它小的有 -10, 10, 11，故 `count[5]=3`。

最终 $b[count[i]] = a[i]$，结果为：**{-10, 10, 11, 19, 25, 25}**。

(2) 比较次数：这是一个标准的两层嵌套循环，外层 0 到 $n-2$，内层 $i+1$ 到 $n-1$。

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

问题 (3) 稳定性分析

稳定性判定：**不稳定**。

- **理由**：观察代码 `if (a[i] < a[j]) count[j]++; else count[i]++;`。当 $a[i] = a[j]$ 时，程序执行 `else` 分支，导致靠前的元素 $a[i]$ 的 `count` 值增加，从而排在靠后的元素 $a[j]$ 之后，颠倒了相对次序。

修改方案：只需将判断相等的逻辑交给靠后的元素，即可保持稳定性。将原代码中的判断逻辑修改为：

```
1     if (a[i] <= a[j]) count[j]++; // 相等时，增加后一个元素的计数
2     else              count[i]++; // 保证前一个元素计数不变，从而排在前面
```

得分关键

1. 模拟时注意两个 25 的先后。2. 比较次数公式要记牢。3. 稳定性的本质是“等值元素相对位置不变”，修改时注意 <= 的使用位置。

2016 年 408 统考真题 - 算法题 (43)

第 43 题

已知集合 A 含有 n 个正整数。设计算法将其划分为 A_1 和 A_2，满足：

- $|n_1 - n_2|$ 最小（即两子集规模尽可能相等）。
- $|S_1 - S_2|$ 最大（即两子集元素和之差尽可能大）。

要求：给出设计思想、代码实现及复杂度分析。

核心目标

若 n 为偶数，$n_1 = n_2 = n/2$；若 n 为奇数，$|n_1 - n_2| = 1$。
要使和之差最大，只需将数组中 **最小的** $\lfloor n/2 \rfloor$ **个数**分给 A_1，其余分给 A_2。

算法设计思想：快速划分法

最优策略：找到数组中下标为 $k = \lfloor n/2 \rfloor$ 的元素，使得左边的元素都小于它，右边的元素都大于它。
算法步骤：

1. 采用**快速排序的划分（Partition）**思想。
2. 选择一个枢轴（Pivot），进行一轮划分。
3. 检查枢轴最终位置 pos：
   - 若 $pos = \lfloor n/2 \rfloor$，则划分完成。
   - 若 $pos < \lfloor n/2 \rfloor$，对右半部分继续划分。
   - 若 $pos > \lfloor n/2 \rfloor$，对左半部分继续划分。
4. 划分结束后，前 $\lfloor n/2 \rfloor$ 个元素即为 A_1，剩余为 A_2。

```
void setPartition(int a[], int n) {
    int low = 0, high = n - 1;
    int k = n / 2; // 划分目标位置
    while (true) {
        int pivot = a[low]; // 选第一个元素为枢轴
        int i = low, j = high;
        while (i < j) { // 标准 Partition 过程
            while (i < j && a[j] >= pivot) j--;
            a[i] = a[j];
            while (i < j && a[i] <= pivot) i++;
            a[j] = a[i];
        }
        a[i] = pivot;
        if (i == k) break; // 恰好找到中位数位置
        else if (i < k) low = i + 1; // 在右半部分找
        else high = i - 1; // 在左半部分找
    } // 此时 a[0...k-1] 为 A1, a[k...n-1] 为 A2
}
```

(3) 时间复杂度和空间复杂度分析

(3) 复杂度分析：

- **时间复杂度**：平均为 $O(n)$。虽然类似快排，但每次只处理一半，满足递归式 $T(n) = T(n/2) + O(n)$。
- **空间复杂度**：$O(1)$。原地划分，仅需常数个辅助变量。

总结

比起直接全排序（$O(n \log n)$），基于快排划分的方法在处理大规模数据时效率更高。

历年真题

1. 线性结构 (11)
2. 树 (4)
3. 图 (10)
4. 排序 + 外排序 (5)
   - 排序 (3)
   - 外排序 (2)
     - 2023-42
     - 2012-41
5. 查找 + 高级查找 (6)

2023 年 408 统考真题 - 外部排序 (42)

第 42 题

对含有 n 个记录的文件进行外部排序，使用置换-选择排序生成初始归并段。工作区能保存 m 个记录。

1. 若 $n = 19$，关键字序列为：51, 94, 37, 92, 14, 63, 15, 99, 48, 56, 23, 60, 31, 17, 43, 8, 90, 166, 100。当 $m = 4$ 时，求生成的初始归并段及其内容。
2. 对任意 m，第一个初始归并段长度的最大值和最小值分别是多少？

置换-选择排序核心逻辑

1. 从工作区选出**最小且不小于**当前归并段最大关键字的记录输出。
2. 若工作区记录均小于当前归并段最大值，则该归并段结束。

问题 (1) 模拟：生成过程解析

工作区 $m = 4$。记当前归并段最后一个输出值为 $LAST$。**归并段 1 (R1):**

- 装入 $\{51, 94, 37, 92\}$，输出 **37** ($LAST = 37$)。
- 补入 14 (< 37 冻结)，输出 **51** ($LAST = 51$)。
- 补入 63，输出 **63** ($LAST = 63$)。
- 补入 15 (< 63 冻结)，输出 **92** ($LAST = 92$)。
- 补入 99，输出 **94** ($LAST = 94$)，补入 48 (冻结)，输出 **99** ($LAST = 99$)。
- 此时工作区全冻结 $\{14, 15, 48, 56\}$。R1 结束。

结果展示：

- **R1:** 37, 51, 63, 92, 94, 99
- **R2:** 14, 15, 23, 31, 43, 48, 56, 60, 90, 100, 166
- **R3:** 8, 17

共生成 **3** 个初始归并段。

问题 (2) 长度极限与要点点评

(2) 第一个初始归并段长度的极限：

- **最小值**：m。当补入的第 $m+1$ 个记录及其后的记录都比当前输出的小时，工作区会迅速填满冻结记录。
- **最大值**：n。当待排序列本身已经有序或接近有序时，第一个归并段可以包含所有记录。

置换-选择排序的“坑”

1. 冻结机制是关键：很多学生会把工作区当成普通的缓冲区。记住：只有“不小于当前段最大值”的最小记录才能输出。
2. 为什么用小根堆？虽然考纲不强制写堆操作，但逻辑上工作区应维护一个小根堆，以保证每次能高效找到最小可用记录。
3. 长度预期：置换-选择排序生成的初始归并段平均长度约为 $2m$，这比传统的内存排序翻了一倍，能有效减少归并趟数。

2012 年 408 统考真题 - 综合应用题 (41)

第 41 题

现有 6 个有序表 $A \ldots F$，元素个数分别为：10, 35, 40, 50, 60, 200。
要求通过 5 次两两合并，将表合并成 1 个升序表，并使最坏情况下比较的总次数达到最小。
问题：

1. 给出完整的合并过程，并求出最坏情况下比较的总次数。
2. 描述 $N(N \geq 2)$ 个不等长升序表的合并策略，并说明理由。

核心原理

两个长度为 m 和 n 的有序表合并，最坏情况下的比较次数为 $m + n - 1$。

合并过程与计算

(1) 合并过程：始终选择当前长度最小的两个表进行合并（贪心策略）。

- **第 1 次**： 10 和 35 合并，得 **45**。比较 $10 + 35 - 1 = 44$ 次。
- **第 2 次**： 40 和 45 合并，得 **85**。比较 $40 + 45 - 1 = 84$ 次。
- **第 3 次**： 50 和 60 合并，得 **110**。比较 $50 + 60 - 1 = 109$ 次。
- **第 4 次**： 85 和 110 合并，得 **195**。比较 $85 + 110 - 1 = 194$ 次。
- **第 5 次**： 195 和 200 合并，得 **395**。比较 $195 + 200 - 1 = 394$ 次。

总比较次数： $44 + 84 + 109 + 194 + 394 = \mathbf{825}$ 次。

通用合并策略与理由

(2) 合并策略：哈夫曼树策略（最佳归并树）
每次从当前序列集合中选取长度最短的两个序列进行合并，直到合并为一个序列为止。
理由：

1. **WPL 的转化**：合并过程中总的比较次数等于归并树的带权路径长度（WPL）减去合并次数（$N-1$）。
2. **最小化原则**：哈夫曼树算法保证了 WPL 最小，即让长度较短的序列位于树的较深层，参与合并的次数更多；长度较长的序列位于较浅层，参与合并的次数更少。
3. **最坏情况优化**：既然每次合并的代价正相关于参与合并的表长之和，通过哈夫曼树构造可以确保总代价（比较次数）达到全局最优。

重点笔记

注意区分：求“最坏情况比较次数”是 $\sum(m+n-1)$；而单纯求“归并树 WPL”是 $\sum(weight \times depth)$。在 408 考试中，一定要减去那 $N-1$ 次比较。

大题分布

1. 线性结构 (11)
2. 树 (4)
3. 图 (10)
4. 排序 + 外排序 (5)
5. 查找 + 高级查找 (6)
   - 哈希/散列表 (2)
   - 二叉搜索树 (2)
   - 二分查找/折半搜索/折半排除 (1)
   - 查找综合

历年真题

1. 线性结构 (11)
2. 树 (4)
3. 图 (10)
4. 排序 + 外排序 (5)
5. 查找 + 高级查找 (6)
   - 哈希/散列表 (2)
     - 2024-42
     - 2010-41
   - 二叉搜索树 (2)
   - 二分查找/折半搜索/折半排除 (1)
   - 查找综合

2024 年 408 统考真题 - 散列表应用 (42)

第 42 题

关键字序列：20, 3, 11, 18, 9, 14, 7。散列表长度 $m = 11$。

- **散列函数**：$H(key) = (key \times 3) \pmod{11}$
- **冲突处理**：二次探测法 $H_k = (H_0 + k^2) \pmod{11}$

任务：(1) 构造散列表并计算装填因子；(2) 查找 14 的比较序列；(3) 查找 8 失败时的散列地址。

核心提醒

本题散列函数并非简单的 $key \pmod{11}$，而是涉及乘法后再取模。二次探测仅使用 k^2 项（即 $1^2, 2^2, \ldots$），不涉及 $-k^2$。

问题 (1): 先逐个计算散列地址：

- **20**: $(20 \times 3) \pmod{11} = 60 \pmod{11} = 5$。**放入 HT[5]**。
- **3**: $(3 \times 3) \pmod{11} = 9$。**放入 HT[9]**。
- **11**: $(11 \times 3) \pmod{11} = 0$。**放入 HT[0]**。
- **18**: $(18 \times 3) \pmod{11} = 54 \pmod{11} = 10$。**放入 HT[10]**。
- **9**: $(9 \times 3) \pmod{11} = 27 \pmod{11} = 5$。**冲突!**
  $\rightarrow H_1 = (5 + 1^2) \pmod{11} = 6$。**放入 HT[6]**。
- **14**: $(14 \times 3) \pmod{11} = 42 \pmod{11} = 9$。**冲突!**
  $\rightarrow H_1 = (9 + 1^2) \pmod{11} = 10$ **(再冲突!)**
  $\rightarrow H_2 = (9 + 2^2) \pmod{11} = 13 \pmod{11} = 2$。**放入 HT[2]**。
- **7**: $(7 \times 3) \pmod{11} = 21 \pmod{11} = 10$。**冲突!**
  $\rightarrow H_1 = (10 + 1^2) \pmod{11} = 0$ **(再冲突!)**
  $\rightarrow H_2 = (10 + 2^2) \pmod{11} = 14 \pmod{11} = 3$。**放入 HT[3]**。

下标	0	1	2	3	4	5	6	7	8	9	10
关键字	11	–	14	7	–	20	9	–	–	3	18

装填因子 $\alpha = 7/11 \approx 0.636$

(2) 查找 14 的关键字比较序列：

1. 计算 $H_0(14) = 9$，比较 HT[9] (**3**)，不匹配。
2. 计算 $H_1 = (9 + 1^2) = 10$，比较 HT[10] (**18**)，不匹配。
3. 计算 $H_2 = (9 + 2^2) \pmod{11} = 2$，比较 HT[2] (**14**)，**命中**！
4. **比较序列**： 3, 18, 14。

(3) 查找 8 失败时的散列地址：

- $H_0(8) = (8 \times 3) \pmod{11} = 24 \pmod{11} = 2$。
- 检查 HT[2] (14)，不为空。
- $H_1(8) = (2 + 1^2) \pmod{11} = 3$。检查 HT[3] (7)，不为空。
- $H_2(8) = (2 + 2^2) \pmod{11} = 6$。检查 HT[6] (9)，不为空。
- $H_3(8) = (2 + 3^2) \pmod{11} = 11 \pmod{11} = 0$。检查 HT[0] (11)，不为空。
- $H_4(8) = (2 + 4^2) \pmod{11} = 18 \pmod{11} = 7$。检查 HT[7] (**空**)。
- **结论**：确认查找失败时的地址是 **7**。

点评

1. **探测终点**：查找失败的标志是遇到“空单元”。在二次探测中，路径可能很长，务必细心计算每一个 k^2。
2. **散列函数陷阱**：很多同学会惯性思维直接用 $8 \pmod{11}$，忽略了题目给定的 $(key \times 3) \pmod{11}$。

2010 年 408 统考真题 - 散列表构造 (41)

第 41 题

关键字序列：$\langle 7, 8, 30, 11, 18, 9, 14 \rangle$。

- **散列函数**：$H(key) = (key \times 3) \pmod 7$
- **冲突处理**：线性探测再散列法
- **装填因子**：$\alpha = 0.7$

任务：(1) 画出散列表；(2) 计算查找成功和失败的 *ASL*。

关键推导：确定表长 m

根据 $\alpha = \frac{n}{m}$，已知 $n = 7, \alpha = 0.7$，得：$m = \frac{7}{0.7} = 10$
因此，散列表是一个下标为 $0 \cdots 9$ 的数组。

问题 (1) 解析：HT 的构造

计算初始散列地址及冲突处理：

- **7**：$(7 \times 3) \pmod 7 = 0$。**放入** HT[0]。
- **8**：$(8 \times 3) \pmod 7 = 3$。**放入** HT[3]。
- **30**：$(30 \times 3) \pmod 7 = 90 \pmod 7 = 6$。**放入** HT[6]。
- **11**：$(11 \times 3) \pmod 7 = 33 \pmod 7 = 5$。**放入** HT[5]。
- **18**：$(18 \times 3) \pmod 7 = 54 \pmod 7 = 5$。**冲突！** $\rightarrow$ HT[6] 满 $\rightarrow$ HT[7]。
- **9**：$(9 \times 3) \pmod 7 = 27 \pmod 7 = 6$。**冲突！** $\rightarrow$ HT[7] 满 $\rightarrow$ HT[8]。
- **14**：$(14 \times 3) \pmod 7 = 0$。**冲突！** $\rightarrow$ HT[1]。

下标	0	1	2	3	4	5	6	7	8	9
关键字	7	14	–	8	–	11	30	18	9	–
探测次数	1	2	–	1	–	1	1	3	3	–

问题 (2) 解析：ASL 计算与点评：

(1) 查找成功时的平均查找长度 ($ASL_{success}$)：只计算表中已有的 7 个元素：

$$ASL_{succ} = \frac{1+2+1+1+1+3+3}{7} = \frac{12}{7} \approx 1.71$$

(2) 查找不成功时的平均查找长度 (ASL_{unsucc})：根据 $H(key) \pmod 7$，只需考察地址 **0-6** 的探测情况：

- 地址 0: 7, 14, 空 (2) $\rightarrow$ 3 次；　地址 1: 14, 空 (2) $\rightarrow$ 2 次
- 地址 2: 空 (2) $\rightarrow$ 1 次；　地址 3: 8, 空 (4) $\rightarrow$ 2 次
- 地址 4: 空 (4) $\rightarrow$ 1 次；　地址 5: 11, 30, 18, 9, 空 (9) $\rightarrow$ 5 次
- 地址 6: 30, 18, 9, 空 (9) $\rightarrow$ 4 次

$$ASL_{unsucc} = \frac{3+2+1+2+1+5+4}{7} = \frac{18}{7} \approx 2.57$$

线性探测

1. **失败长度的“分母”**：很多同学会除以表长 10。注意：ASL_{unsucc} 的分母应与散列函数取模的值保持一致。
2. **探测深度**：查找不成功是指从初始散列地址开始，一直探测到“空单元”为止的比较次数。

历年真题

1. 线性结构 (11)
2. 树 (4)
3. 图 (10)
4. 排序 + 外排序 (5)
5. 查找 + 高级查找 (6)
   - 哈希/散列表 (2)
   - 二叉搜索树 (2)
     - 2026-41
     - 2022-41
   - 二分查找/折半搜索/折半排除 (1)
   - 查找综合

第 41 题

已知 BST 采用二叉链表存储。给定整数 K，查找树中关键字与 K 之差的绝对值最小的所有结点。

- **要求**：给出设计思想、代码实现。
- **核心**：需要处理可能存在多个结点（如 $K = 10$，树中有 9 和 11）的情况。

关键性质

在 BST 中，离 K 最近的结点一定位于：1. 路径上与 K 相等的结点；2. 若不存在 K，则是 K 插入路径上的某个祖先或其前驱/后继。

算法设计思想：动态维护最小差值

基本思想：利用 BST 的查找逻辑，从根节点开始向下搜索。

1. **初始化**：设全局变量 `minDiff` 为正无穷，用于记录当前发现的最小绝对差。
2. **遍历过程**：
   - 计算当前结点 p 与 K 的差值 $diff = |p \to data - K|$。
   - **若** $diff < minDiff$：更新 `minDiff`，清空之前保存的结果，记录当前关键字。
   - **若** $diff == minDiff$：将当前关键字加入结果列表。
3. **分支选择**：
   - 若 $p \to data == K$：说明最小差值为 0，查找结束。
   - 若 $p \to data > K$：最小差值可能在左子树中，向左走。
   - 若 $p \to data < K$：最小差值可能在右子树中，向右走。

```c
int minDiff = 0x7fffffff; // 初始为最大整数
int results[MAX_SIZE], count = 0; // 记录满足条件的关键字

void findMinAbs(BTNode *T, int K) {
    BTNode *p = T;
    while (p != NULL) {
        int currentDiff = abs(p->data - K);
        if (currentDiff < minDiff) {
            minDiff = currentDiff; // 发现更小的差值
            count = 0;             // 重置结果集
            results[count++] = p->data;
        } else if (currentDiff == minDiff) {
            results[count++] = p->data; // 差值相等，并列存入
        }

        if (p->data == K) break; // 差值为0，最优解已得
        else if (p->data > K) p = p->left;  // 目标可能在左侧
        else p = p->right;                  // 目标可能在右侧
    }
    // 输出结果
    printf("最小绝对差为：%d\n", minDiff);
    for (int i = 0; i < count; i++) printf("%d ", results[i]);
}
```

1. 为什么不用中序遍历？

虽然中序遍历可以得到有序序列，进而找到 K 的前驱和后继，但其时间复杂度为 $O(n)$。利用 BST 查找性质可以将时间降低到 $O(h)$（h 为树高），这才是考场要求的高效算法。

2. 陷阱：多个解的处理

题目要求输出“所有结点”。在搜索过程中，必须考虑 $|p \to data - K| == minDiff$ 的情况。例如 $K = 15$，树中有 14 和 16，这两个点都是答案。

3. 复杂度总结

- **时间复杂度**：$O(h)$，平均 $O(\log n)$，最坏 $O(n)$。
- **空间复杂度**：$O(1)$（若使用递归则为 $O(h)$，此例使用非递归）。

2022 年 408 统考真题 - 算法题 (41)

第 41 题

非空二叉树 T 采用顺序存储（数组 `SqBiTNode`），不存在的结点用 -1 表示。
设计一个高效算法：判定该二叉树是否为**二叉搜索树 (BST)**。

关键知识点：顺序存储下标关系

若根结点下标为 0：

- 左孩子下标：$2i+1$
- 右孩子下标：$2i+2$
- 判定条件：下标需 $<$ *MAX_SIZE* 且值为正整数（$\neq -1$）。

算法设计思想：中序遍历有序性

核心思想：二叉搜索树的中序遍历序列应当是一个严格递增的序列。
具体步骤：

1. 对顺序存储的二叉树进行**中序遍历**。由于是数组存储，递归访问时根据下标关系跳转。
2. **记录前驱**：使用一个静态变量或全局变量 `pre` 记录上一个访问的结点值。
3. **比较判定**：每访问一个有效结点，将其与 `pre` 比较。若当前值 $\leq$ `pre`，则不满足递增特性，判定非 BST。
4. **剪枝优化**：一旦发现非递增，立即停止递归并返回 `false`。

```
int pre = -1; // 全局变量，记录前驱结点值
bool isBST(SqBiTree T, int index) {
    if (index >= T.ElemNum || T.SqBiTNode[index] == -1)
        return true; // 递归边界：索引越界或空结点
    // 1. 递归判定左子树
    if (!isBST(T, 2 * index + 1)) return false;
    // 2. 访问当前结点
    if (T.SqBiTNode[index] <= pre) return false; // 非递增则返回false
    pre = T.SqBiTNode[index]; // 更新前驱值
    // 3. 递归判定右子树
    return isBST(T, 2 * index + 2);
}
// 调用入口：isBST(T, 0);
```

复杂度分析

时间复杂度：$O(n)$，每个有效结点仅访问一次。
空间复杂度：$O(h)$，h 为树的高度（递归栈深度）。

1. 为什么不能只比“父子”？

陷阱：仅判定“左孩 $<$ 父 $<$ 右孩”是不够的。BST 要求左子树**所有**结点均小于根。中序遍历通过维护全局序，完美解决了这一全局性限制。

2. 顺序存储的边界处理

题目中给出的 `ElemNum` 是“实际占用的数组元素个数”。在访问 $2i+1$ 或 $2i+2$ 时，必须判断其是否超出了 `ElemNum` 的范围，否则会导致非法内存访问。

3. 评分标准解读

408 阅卷中，使用递归中序遍历是标准解法。如果使用非递归（栈）同样可行，但逻辑较复杂，建议优先使用递归以节省考场时间。

历年真题

1. 线性结构 (11)
2. 树 (4)
3. 图 (10)
4. 排序 + 外排序 (5)
5. 查找 + 高级查找 (6)
   - 哈希/散列表 (2)
   - 二叉搜索树 (2)
   - 二分查找/折半搜索/折半排除 (1)
     - 2011-42
   - 查找综合

2011 年 408 统考真题 - 算法题 (42)

第 42 题

两个等长升序序列 A 和 B，长度均为 n。找出两个序列合并后的中位数（即第 n 个小的元素）。
示例：$S_1 = (11, 13, 15, 17, 19)$，$S_2 = (2, 4, 6, 8, 20)$
合并序列为 $(2, 4, 6, 8, 11, 13, 15, 17, 19, 20)$，**中位数为 11**。

效率目标

- **常规方案**：合并排序，$O(n)$ 时间。
- **最优方案**：模拟二分查找，$O(\log n)$ 时间。

算法设计思想：折半排除法

分别求出 A 和 B 的中位数 m_1 和 m_2：

1. **若** $m_1 = m_2$：则该值即为两序列的中位数。
2. **若** $m_1 < m_2$：中位数必在 m_1 的右侧和 m_2 的左侧。此时舍弃 A 的前半部分和 B 的后半部分（保持舍弃长度相等）。
3. **若** $m_1 > m_2$：中位数必在 m_1 的左侧和 m_2 的右侧。此时舍弃 A 的后半部分和 B 的前半部分。
4. **重复循环**：直到两个序列均只剩一个元素，较小者即为所求。

注意细节

舍弃时需保证两序列长度同步减小。若元素个数为奇数，舍弃前后半部分时要包含中位数；若为偶数，则不包含。

```c
int M_Search(int A[], int B[], int n) {
    int s1 = 0, d1 = n - 1, m1;
    int s2 = 0, d2 = n - 1, m2;
    while (s1 != d1 || s2 != d2) {
        m1 = (s1 + d1) / 2;
        m2 = (s2 + d2) / 2;
        if (A[m1] == B[m2]) return A[m1]; // 情况1
        if (A[m1] < B[m2]) { // 情况2：舍弃 A 前半和 B 后半
            if ((s1 + d1) % 2 == 0) { // 元素个数为奇数
                s1 = m1; d2 = m2;
            } else { // 元素个数为偶数
                s1 = m1 + 1; d2 = m2;
            }
        } else { // 情况3：舍弃 A 后半和 B 前半
            if ((s1 + d1) % 2 == 0) { // 元素个数为奇数
                d1 = m1; s2 = m2;
            } else { // 元素个数为偶数
                d1 = m1; s2 = m2 + 1;
            }
        }
    }
    return A[s1] < B[s2] ? A[s1] : B[s2];
}
```

(3) 复杂度分析

(3) 复杂度分析：

- **时间复杂度**： $O(\log n)$。每次循环规模减半。
- **空间复杂度**： $O(1)$。仅需若干指针变量。

1. 为什么是 $O(\log n)$？

本质上是在两个有序数组中寻找**第 k 小的数**（此处 $k = n$）。利用有序性，每轮比较中位数后，可以确定性地排除掉一半不可能的候选者。这种“减而治之”的思想是二分查找的延伸。

2. 易错点：边界控制

许多考生能想到二分，但难以拿满分，原因在于**奇偶长度的索引处理**：

- **奇数长度**：舍弃一半后，中位数必须保留在剩余区间内。
- **偶数长度**：舍弃一半时，中位数位置的取舍必须保持两数组剩余长度严格相等。

3. 考场备选方案（保底策略）

如果你在考场上无法完美写出 $O(\log n)$ 的二分逻辑，应果断编写**双指针合并法**：

- 逻辑：使用两个指针同时遍历 A, B，比较并计数，直到找到第 n 个数。
- 评价：时间 $O(n)$，空间 $O(1)$。虽不是最优解，但逻辑简单易对，通常能拿到 10–12 分（满分 15）。

历年真题

1. 线性结构 (11)
2. 树 (4)
3. 图 (10)
4. 排序 + 外排序 (5)
5. 查找 + 高级查找 (6)
   - 哈希/散列表 (2)
   - 二叉搜索树 (2)
   - 二分查找/折半搜索/折半排除 (1)
   - 查找综合
     - 2013-42

2013 年 408 统考真题 - 查找效率分析 (42)

第 42 题

集合 $S = \{\text{do, for, repeat, while}\}$，**查找概率为**：$p_1 = 0.35, p_2 = 0.15, p_3 = 0.15, p_4 = 0.35$。

- **现状**：长度为 4 的顺序表，折半查找，$ASL = 2.2$。
- **任务**：(1) 顺序存储下如何排列以获得更短 ASL？采用何种方法？计算 ASL。(2) 链式存储下如何排列以获得更短 ASL？采用何种方法？计算 ASL。

核心原理：哈夫曼思想在查找中的应用

要使总查找代价最小，应遵循：**高概率元素放在搜索路径更短（比较次数更少）的位置**。

问题 (1) 解析：顺序存储结构优化

在顺序存储中，折半查找的比较次数由判定树决定。

1. **判定树结构**： 4 个元素的折半查找判定树，根节点比较 1 次，第二层 2 个节点比较 2 次，第三层 1 个节点比较 3 次。
2. **优化策略**：将概率最大的两个元素（0.35）放在第一层（根）和第二层。
3. **排列方式**：假设顺序表为 (a_1, a_2, a_3, a_4)，折半查找过程：先看 $\lfloor (1+4)/2 \rfloor = 2$。根是 a_2。再看左子树 a_1，右子树 $\lfloor (3+4)/2 \rfloor = 3$。$a_4$ 在第三层。**结论**：应将高概率元素放在 a_2 或 a_3 位置。例如：**(for, do, while, repeat)**。
4. **ASL 计算**： $ASL = 0.35 \times 1 + (0.35 + 0.15) \times 2 + 0.15 \times 3 = \mathbf{1.8}$。（注：比原先 2.2 显著降低）

问题 (2) 解析：链式存储结构优化

链式存储不支持随机访问，因此无法高效进行折半查找。

1. **查找方法**： **顺序查找**（从头指针开始逐个遍历）。
2. **优化策略**： **按概率降序排列**。将概率最大的元素放在链表最前端。
3. **排列方式**： **(do, while, for, repeat) 或 (while, do, repeat, for)**。即：0.35, 0.35, 0.15, 0.15。
4. **ASL 计算**： $ASL = \sum(p_i \times \text{查找次数}_i)$
   $ASL = 0.35 \times 1 + 0.35 \times 2 + 0.15 \times 3 + 0.15 \times 4$ $ASL = 0.35 + 0.7 + 0.45 + 0.6 = \mathbf{2.1}$。

1. 概率与位置的映射

无论何种数据结构，优化的通用准则都是：**让频繁访问的元素更早被找到**。在树形查找中，它们应靠近根；在线性查找中，它们应靠近起点。

2. 容易忽略的折半查找前提

折半查找要求必须是**顺序存储**且**关键字有序**。本题中元素是字符串，如果为了降低 ASL 调整了排列顺序，实际上是改变了查找表中“关键字”定义的顺序，使其符合新的折半判定树。

3. 延伸思考

如果概率差异极大，最优的顺序存储结构可能不是折半查找，而是构造一棵**最优二叉查找树 (Optimal BST)**，其原理与哈夫曼树类似，但保留了二叉排序树的性质。

欢迎提问！

你的疑惑，我的动力