算法设计与分析第三次作业

姓名：王泽黎
学号：2022K8009929011

1. Distributing Candy Game

1.1 Modeling

核心要求：
1. n个孩子和1个老师，每人左右手各写一个整数
2. 老师必须在队伍最前面
3. 每个孩子获得的糖果数 = floor(前面所有人左手数的乘积 / 自己右手的数)
4. 目标：通过重新排列孩子的顺序，使得获得最多糖果的孩子的糖果数最小

关键变量：
1. n: 孩子的数量
2. teacher_left: 老师左手的数
3. teacher_right: 老师右手的数
4. children[i].left: 第i个孩子左手的数
5. children[i].right: 第i个孩子右手的数
6. current_order[]: 当前排列顺序
7. min_max_candy: 最优解下的最大糖果数

分析思路：
1. 每个孩子获得的糖果数取决于其在队列中的位置，前面的人左手数越大，后面的人获得的糖果就越多，自己右手的数越大，获得的糖果就越少
2. 对于每个位置i，前面所有人左手数的乘积都会影响到后面所有人，右手数大的孩子应该尽量放在后面，这样可以减少他们获得的糖果，左手数大的孩子放在前面会导致后面的人获得更多糖果
3. 可以考虑使用贪心策略或动态规划

1.2 Algorithm description

1. 初始化变量
    - 定义二维数组 dp，dp[mask][last] 表示当前状态为 mask，最后一个孩子的位置为 last 时，获得最多糖果的孩子的糖果数的最小值

- dp

$$0$$

$$0$$

= 0，表示没有孩子时，糖果数为 0，其他状态初始化为无穷大

2. 状态转移
   - 对于状态 dp[mask][last]：
     - 枚举下一个可以放置的孩子 next
     - 计算放置 next 后该孩子获得的糖果数
     - 转移方程：dp[new_mask][next] = min(dp[new_mask][next], max(dp[mask][last], floor(prod / children[next].right)))
   - 按照mask的二进制中1的个数从小到大枚举
3. 返回结果
   - 返回 dp[(1 << n) - 1][n]

1.3 Complexity analysis

- 时间复杂度：O(n^2 * 2^n)，时间复杂度主要需要考虑计算 dp 数组的过程，需要遍历 n 个孩子，每个孩子有 2^n 种状态
- 空间复杂度：O(n * 2^n)，空间复杂度主要需要考虑 dp 数组的空间占用

2. Array Partition

2.1 Modeling

核心要求：
1. 输入是长度为2n的整数数组
2. 需要将数组分成n对数字
3. 对每一对取较小值
4. 所有较小值的和要最大

关键变量：
1. n: 数组长度的一半，表示要形成的数对数量
2. nums[]: 输入数组
3. sum: 所有数对中较小值的和

分析思路：
1. 每对数字中较大的数不会影响最终结果，为了最大化和，应该尽量让每对中的较小值尽可能大

2. 将数组排序后，从小到大每两个数作为一对，较大的数两两配对，避免浪费
3. 正确性：假设存在更优的方案，那么必然存在交叉配对的情况，但尝试交换配对中的两个数，会使得较小值更小，因此原方案是最优的

2.2 Algorithm description

1. 初始化变量
    - 对数组 nums 排序
2. 计算和
    - 遍历数组 nums，每次取两个数中的较小值，累加到 sum 中
3. 返回结果

2.3 Complexity analysis

- 时间复杂度：O(nlogn)，时间复杂度主要需要考虑排序的过程
- 空间复杂度：O(1)，只需要常数级别的额外空间

3. Delete Number Game

3.1 Modeling

核心要求：
1. 输入一个长度为n的非负整数（可能有前导零）
2. 删除其中k个数字（k < n）
3. 剩余数字按原顺序组成新数（可以有前导零）
4. 目标：使新数最小

关键变量：
1. n: 原数字长度
2. k: 需要删除的数字个数
3. num: 输入的数字字符串
4. result: 存储结果的栈/数组
5. remain: 还需要删除的数字个数

分析思路：
1. 对于相邻的两个数字，如果前面的数字大于后面的数字，删除前面的数字会使结果更小
2. 考虑贪心策略，从左到右遍历数字，当前数字比栈顶小时，不断弹出栈顶（直到k个删完或栈空），如果遍历完还没删够k个数，从栈尾继续删除，并且删除结果前导零

3.2 Algorithm description

1. 初始化变量
    - 定义栈 result，用于存储结果
    - 定义 remain，表示还需要删除的数字个数
2. 遍历数字
    - 对于每个数字 num[i]：
        - 如果栈不为空且栈顶大于当前数字，弹出栈顶，remain--
        - 将当前数字入栈
    - 如果遍历完后还有剩余数字，从栈尾删除，删除结果前导零
3. 返回结果

3.3 Complexity analysis

- 时间复杂度：O(n)，时间复杂度主要需要考虑遍历数字的过程
- 空间复杂度：O(n)，空间复杂度主要需要考虑栈 result 的空间占用

4. Container Balancing Operations

4.1 Modeling

核心要求：
1. n个容器排成一行，每个容器有一定数量的物品
2. 每次操作可以选择任意数量的容器，每个选中的容器向相邻容器移动1个物品
3. 求使所有容器物品数相等的最小操作次数
4. 如果无法实现平衡，返回-1

关键变量：
1. n: 容器数量
2. containers[]: 初始每个容器的物品数量
3. total: 总物品数
4. target: 目标每个容器的物品数
5. operations: 最少操作次数

分析思路：
1. 计算总物品数total，如果total不能被n整除，返回-1
2. 每次移动物品只能在相邻容器之间进行，对于任意位置i，从左侧累积到i的物品总和应该是i*target，差值表示需要通过位置i的物品流动次数

4.2 Algorithm description

1. 初始化变量
    - 计算总物品数 total
    - 计算目标每个容器的物品数 target
2. 判断是否能平衡
    - 如果 total 不能被 n 整除，返回 -1
3. 计算操作次数
    - 遍历容器 containers，计算每个位置 i 的物品总数 sum
    - 计算 sum - i * target 的绝对值，更新最大值为 operations
4. 返回结果

4.3 Complexity analysis

- 时间复杂度：O(n)，时间复杂度主要需要考虑遍历容器的过程
- 空间复杂度：O(1)，只需要常数级别的额外空间