

《算法设计与分析》——动态规划专题特训

桓

2025 年 6 月 25 日

第一部分：线性/序列动态规划

问题 1：最长递增子序列 (LIS)

给定一个无序的整数数组 'nums'，请找出其中最长递增子序列的长度。子序列可以不连续。例如，对于数组 '[10, 9, 2, 5, 3, 7, 101, 18]'，其最长递增子序列为 '[2, 3, 7, 101]'，因此长度为 4。

要求：

- 1) **状态定义：**定义目标函数（DP 数组）' $dp[i]$ ' 的确切物理含义。
- 2) **递推关系：**写出状态转移方程。
- 3) **初值：**说明 DP 数组的初始值。
- 4) **算法伪代码：**给出基于动态规划求解该问题的伪代码。

问题 2：俄罗斯套娃信封问题

给定一些信封，每个信封有两个整数 ' (w, h) ' 分别表示信封的宽度和高度。当一个信封的宽度和高度都比另一个信封大时，这个信封就可以放进另一个信封里。请计算最多能有多少个信封能组成一组“俄罗斯套娃”信封（即一个套一个）。

要求：

- 1) **问题转化：**说明如何将此问题转化为一个经典的动态规划模型。
- 2) **状态定义与递推关系：**基于转化后的模型，给出状态定义和状态转移方程。
- 3) **算法伪代码：**描述完整的算法流程。
- 4) **复杂度分析：**分析算法的时间和空间复杂度。

问题 3：最长公共子序列 (LCS)

给定两个字符串 ' $text1$ ' 和 ' $text2$ '，返回这两个字符串的最长公共子序列的长度。如果不存在公共子序列，则返回 0。

要求：

- 1) **状态定义：**定义二维 DP 数组 ' $dp[i][j]$ ' 的确切物理含义。

- 2) **递推关系**: 写出状态转移方程, 需要考虑 `'text1[i-1] == text2[j-1]'` 和 `'text1[i-1] != text2[j-1]'` 两种情况。
- 3) **初值**: 说明 DP 数组的初始化过程。
- 4) **算法伪代码**: 给出求解该问题的伪代码。

问题 4: 编辑距离

给定两个单词 `'word1'` 和 `'word2'`, 请计算将 `'word1'` 转换成 `'word2'` 所使用的最少操作数。你可以对一个单词进行如下三种操作: 插入一个字符、删除一个字符、替换一个字符。

要求:

- 1) **状态定义**: 定义二维 DP 数组 `'dp[i][j]'` 的确切物理含义。
- 2) **递推关系**: 写出状态转移方程, 需要清晰地对应三种操作。
- 3) **初值**: 说明 DP 数组的边界条件 (例如 `'dp[i][0]'` 和 `'dp[0][j]'`) 如何初始化。
- 4) **算法伪代码**: 给出求解该问题的伪代码。

问题 5: 打家劫舍 (House Robber) - 环形数组

你是一个专业的小偷, 计划偷窃一个环形街道上的房屋。每间房内都藏有一定的现金, 影响你偷窃的唯一制约因素就是相邻的房屋装有相互连通的防盗系统, 如果两间相邻的房屋在同一晚上被小偷闯入, 系统会自动报警。给定一个代表每间房屋存放金额的非负整数数组 `'nums'`, 计算你在不触动警报装置的情况下, 能够偷窃到的最高金额。

要求:

- 1) **问题分解**: 说明如何将环形数组问题分解为两个或多个线性数组问题。
- 2) **状态定义与递推关系**: 针对线性数组问题, 给出 DP 的状态定义和转移方程。
- 3) **算法伪代码**: 描述解决原始环形数组问题的完整算法。

问题 6: 单词拆分 (Word Break)

给定一个非空字符串 `'s'` 和一个包含非空单词列表的字典 `'wordDict'`, 判定 `'s'` 是否可以被空格拆分为一个或多个在字典中出现的单词。例如, `'s = "leetcode"'`, `'wordDict = ["leet", "code"]'`, 返回 `'true'`。

要求:

- 1) **状态定义**: 定义 DP 数组 `'dp[i]'` 的确切物理含义。
- 2) **递推关系**: 写出状态转移方程。
- 3) **初值**: 说明 DP 数组的初始值。
- 4) **算法伪代码**: 给出求解该问题的伪代码。

问题 7：买卖股票的最佳时机含冷冻期

给定一个整数数组 'prices'，其中 'prices[i]' 表示某支股票在第 'i' 天的价格。设计一个算法来计算你能获取的最大利润。你最多可以完成无数次交易，但有以下限制：卖出股票后，你无法在第二天买入股票（即存在 1 天的冷冻期）。

要求：

- 1) **状态定义：**设计一个 DP 状态表示。由于状态复杂，可能需要多个 DP 数组或一个多维 DP 数组来表示“持有股票”、“不持有股票且在冷冻期”、“不持有股票且不在冷冻期”等状态。请清晰定义每一个状态。
- 2) **递推关系：**写出不同状态之间的转移方程。
- 3) **初值：**说明各 DP 状态的初始值。
- 4) **算法伪代码：**给出求解该问题的伪代码。

问题 8：解码方法

一条包含字母 A-Z 的消息通过以下方式进行了编码：'A' -> 1, 'B' -> 2, ..., 'Z' -> 26。给定一个只包含数字的非空字符串 's'，请计算解码方法的总数。例如，'s = "226"'，它可以解码为"BZ" (2 26)，"VF" (22 6)，或者"BBF" (2 2 6)，所以答案是 3。

要求：

- 1) **状态定义：**定义 DP 数组 'dp[i]' 的确切物理含义。
- 2) **递推关系：**写出状态转移方程，需要仔细考虑一位解码和两位解码的条件。
- 3) **初值：**说明 'dp[0]' 和 'dp[1]' 的初始化。
- 4) **算法伪代码：**给出求解该问题的伪代码。

问题 9：最长有效括号

给定一个只包含 '(' 和 ')' 的字符串，找出最长有效（格式正确且连续）括号子串的长度。例如，对于 's = "()" '，最长有效括号子串是 "()" '，长度为 2。对于 's = ")()()" '，最长有效括号子串是 "()()" '，长度为 4。

要求：

- 1) **状态定义：**定义 DP 数组 'dp[i]' 的确切物理含义。
- 2) **递推关系：**写出状态转移方程。这可能需要分情况讨论 's[i]' 是 '(' 还是 ')'，以及 's[i-1]' 的情况。
- 3) **初值：**说明 DP 数组的初始化。
- 4) **算法伪代码：**给出求解该问题的伪代码。

问题 10: 粉刷房子 II

假如有一排 ‘n’ 个房子，共 ‘k’ 种颜色。每个房子可以用 ‘k’ 种颜色中的一种来粉刷，不同房子粉刷不同颜色的成本不同。成本由一个 ‘n x k’ 的矩阵 ‘costs’ 表示。要求粉刷所有房子，且相邻的两个房子颜色不能相同，求最小的总成本。

要求：

- 1) **状态定义：**定义 DP 数组 ‘dp[i][j]’ 的含义。
- 2) **递推关系：**写出状态转移方程。请注意，朴素的转移 ‘O(k)’，总复杂度为 ‘O(nk²)’。
- 3) **优化：**描述如何优化状态转移过程，将每次转移的复杂度从 ‘O(k)’ 降到 ‘O(1)’，从而使总时间复杂度达到 ‘O(nk)’。
- 4) **算法伪代码：**给出优化后的算法伪代码。

第二部分：区间动态规划

问题 11: 矩阵链乘法

给定 n 个矩阵的序列 $\langle A_1, A_2, \dots, A_n \rangle$ ，我们希望计算它们的乘积 $A_1 A_2 \dots A_n$ 。矩阵乘法满足结合律，因此任何加括号的方式都会得到同样的结果。但是，不同的加括号方式会导致计算总代价（标量乘法次数）的巨大差异。给定一个数组 ‘p’，其中 ‘p[i-1]’ 和 ‘p[i]’ 分别是矩阵 A_i 的行数和列数，求最小的计算代价。

要求：

- 1) **状态定义：**定义二维 DP 数组 ‘m[i][j]’ 的确切物理含义。
- 2) **递推关系：**写出 ‘m[i][j]’ 的状态转移方程。
- 3) **初值：**说明 DP 数组的初始值。
- 4) **算法伪代码：**给出求解该问题的伪代码，并说明循环的顺序。

问题 12: 戳气球 (Burst Balloons)

有 ‘n’ 个气球，编号为 ‘0’ 到 ‘n-1’，每个气球上都标有一个数字，这些数字存在数组 ‘nums’ 中。现在要求你戳破所有的气球。每当你戳破一个气球 ‘i’ 时，你可以获得 ‘nums[left] * nums[i] * nums[right]’ 个硬币。这里的 ‘left’ 和 ‘right’ 代表和 ‘i’ 相邻的两个气球的序号。当一个气球被戳破后，它左右两边的气球就变成了相邻的气球。求所能获得硬币的最大数量。（你可以假设 ‘nums[-1] = nums[n] = 1’，但它们不能被戳破。）

要求：

- 1) **核心思想：**这个问题直接思考先戳哪个气球很困难。请说明如何通过“逆向思维”（即考虑最后戳破哪个气球）来构建 DP 模型。
- 2) **状态定义：**定义二维 DP 数组 ‘dp[i][j]’ 的确切物理含义。
- 3) **递推关系：**写出状态转移方程。
- 4) **算法伪代码：**给出求解该问题的伪代码。

问题 13: 石子游戏 (Stone Game)

Alex 和 Lee 用几堆石子在做游戏。偶数堆石子 'piles' 排成一行，每堆都有正整数颗石子 'piles[i]'。游戏以谁手中的石子最多来决出胜负。总石子数为奇数，所以没有平局。Alex 和 Lee 轮流进行，Alex 先开始。每次玩家可以从行的开头或结尾取走整堆石头。这个过程一直持续到没有更多的石子堆为止。假设两人都采取最优策略，请判断 Alex 是否能赢得比赛。

要求：

- 1) **状态定义：**定义二维 DP 数组 'dp[i][j]' 的含义。它可以表示在区间 '[i,j]' 上，先手玩家相对于后手玩家能多拿到的最大石子数。
- 2) **递推关系：**写出状态转移方程。
- 3) **最终判断：**说明如何根据 DP 的最终结果来判断 Alex 是否获胜。
- 4) **算法伪代码：**给出求解该问题的伪代码。

问题 14: 奇怪的打印机 (Strange Printer)

有一台奇怪的打印机，它有以下两个特点：1. 打印机每次只能打印同一个字符序列。2. 在任意位置 '(x, y)'，它可以将 'str[x]' 到 'str[y]' 的所有字符都修改成同一个字符。每次操作算作一次。给定一个字符串，计算打印出这个字符串所需的最少操作次数。

要求：

- 1) **状态定义：**定义二维 DP 数组 'dp[i][j]' 的确切物理含义。
- 2) **递推关系：**写出状态转移方程。这需要考虑如何通过分割区间 '[i,j]' 来完成打印。
- 3) **初值与遍历顺序：**说明 DP 数组的初始化和计算的遍历顺序。
- 4) **算法伪代码：**给出求解该问题的伪代码。

问题 15: 给表达式添加运算符

给定一个仅包含数字 '0-9' 的字符串 'num' 和一个目标值 'target'，在 'num' 的数字之间添加 '+', '-', '*' 运算符，使得表达式的结果等于 'target'。返回所有可能的表达式。

要求：这个问题通常使用回溯法解决。但请你尝试从动态规划的角度思考。

- 1) **状态定义：**尝试定义一个 DP 状态。这个状态可能比较复杂，例如 'dp[i]' 可以是一个哈希表或列表，存储所有由 'num[0...i]' 构成的表达式及其计算结果。
- 2) **递推关系：**描述如何从 'dp[j]' (其中 'j<i') 推导出 'dp[i]' 的状态。
- 3) **挑战分析：**分析为什么纯粹的动态规划方法（尤其是只存储数值结果）处理乘法时会遇到困难，以及为什么回溯法在这里更自然。

第三部分：背包问题

问题 16：0-1 背包问题

有 N 件物品和一个容量为 V 的背包。第 i 件物品的体积是 $v[i]$ ，价值是 $w[i]$ 。求解将哪些物品装入背包，可使这些物品的总体积不超过背包容量，且总价值最大。

要求：

- 1) **状态定义：**定义二维 DP 数组 $dp[i][j]$ 的确切物理含义。
- 2) **递推关系：**写出状态转移方程。
- 3) **空间优化：**说明如何将空间复杂度从 $O(NV)$ 优化到 $O(V)$ ，并写出优化后的伪代码。

问题 17：分割等和子集

给定一个只包含正整数的非空数组 $nums$ 。请判断是否可以将这个数组分割成两个子集，使得两个子集的元素和相等。

要求：

- 1) **问题转化：**说明如何将此问题转化为一个 0-1 背包问题。
- 2) **状态定义与递推关系：**基于转化后的模型，给出 DP 的状态定义和转移方程。
- 3) **算法伪代码：**给出求解该问题的伪代码。

问题 18：完全背包问题（零钱兑换）

给定不同面额的硬币 $coins$ 和一个总金额 $amount$ 。编写一个函数来计算可以凑成总金额所需的最少的硬币个数。如果没有任何一种硬币组合能组成总金额，返回 -1。你可以认为每种硬币的数量是无限的。

要求：

- 1) **状态定义：**定义 DP 数组 $dp[i]$ 的确切物理含义。
- 2) **递推关系：**写出状态转移方程。
- 3) **初值：**说明 DP 数组的初始化（特别是如何表示“无法凑成”的状态）。
- 4) **算法伪代码：**给出求解该问题的伪代码。

问题 19：组合总和 IV

给你一个由不同整数组成的数组 $nums$ ，和一个目标整数 $target$ 。请你从 $nums$ 中找出并返回总和为 $target$ 的元素组合的个数。题目数据保证答案符合 32 位整数范围。注意，顺序不同的序列被视作不同的组合。

要求：

- 1) **问题分析：**分析此问题与经典组合问题（不考虑顺序）的区别，并说明为什么它是一个“排列”问题，适合用完全背包的思路解决。

- 2) **状态定义与递推关系**: 给出 DP 的状态定义和转移方程。
- 3) **遍历顺序**: 解释为什么此问题的内外循环顺序（先遍历背包容量还是先遍历物品）至关重要，并说明正确的顺序及其原因。
- 4) **算法伪代码**: 给出求解该问题的伪代码。

问题 20: 目标和 (Target Sum)

给定一个非负整数数组 'nums' 和一个目标数 'S'。现在你有两个符号 '+' 和 '-'。对于数组中的任意一个整数，你都可以从 '+' 或 '-' 中选择一个符号添加在前面。返回可以使最终数组和为 'S' 的所有添加符号的方法数。

要求:

- 1) **问题转化**: 将原问题转化为一个寻找子集和的问题。推导出子集和 'P' 与原数组总和 'sum'、目标 'S' 之间的关系。
- 2) **模型识别**: 指出转化后的问题属于哪种背包模型。
- 3) **状态定义与递推关系**: 基于转化后的模型，给出 DP 的状态定义和转移方程。
- 4) **算法伪代码**: 给出求解该问题的伪代码。

问题 21: 多重背包问题

有 'N' 种物品和一个容量为 'V' 的背包。第 'i' 种物品最多有 'c[i]' 件，每件体积是 'v[i]'，价值是 'w[i]'。求解将哪些物品装入背包，可使这些物品的总体积不超过背包容量，且总价值最大。

要求:

- 1) **朴素解法**: 描述如何将多重背包问题转化为 0-1 背包问题进行求解，并分析其时间复杂度。
- 2) **二进制优化**: 详细描述如何使用二进制分组的思想，将每种物品拆分成 ' $\log(c[i])$ ' 件新的物品，从而优化转化过程。
- 3) **状态定义与递推关系**: 给出优化后的 0-1 背包问题的 DP 定义和递推。
- 4) **复杂度分析**: 分析二进制优化后算法的时间复杂度。

第四部分: 树形/状态压缩/数位 DP 及其他

问题 22: 打家劫舍 III (Tree DP)

一个小偷发现了一个新的可行窃的地区。这个地区只有一个入口，称之为“根”。除了“根”之外，每栋房子有且只有一个“父”房子与之相连。一番侦察之后，聪明的小偷意识到“这个地方的所有房屋的排列类似于一棵二叉树”。如果两个直接相连的房子在同一天晚上被闯入，将自动报警。计算小偷能在不触动警报的情况下，从整个社区中偷窃到的最大金额。

要求:

- 1) **状态定义**: 定义在树的每个节点 'node' 上的 DP 状态。由于选择偷或不偷当前节点会影响子节点的选择, 因此每个节点需要返回一个包含两种状态信息的数组或元组, 例如 '[偷此节点的最大收益, 不偷此节点的最大收益]'。
- 2) **递推关系**: 基于后序遍历的思想, 写出如何根据子节点的 DP 状态计算当前节点的 DP 状态。
- 3) **算法伪代码**: 给出求解该问题的伪代码。

问题 23: 最小路径和

给定一个包含非负整数的 ' $m \times n$ ' 网格 'grid', 请找出一条从左上角到右下角的路径, 使得路径上的数字总和为最小。说明: 每次只能向下或者向右移动一步。

要求: 这是一个基础的二维 DP 问题, 请用它来展示你对 DP 基本要素的掌握。

- 1) **状态定义**: 定义二维 DP 数组 ' $dp[i][j]$ ' 的确切物理含义。
- 2) **递推关系**: 写出状态转移方程。
- 3) **初值**: 说明 DP 数组的边界条件 (第一行和第一列) 如何初始化。
- 4) **算法伪代码**: 给出求解该问题的伪代码。

问题 24: 最大正方形

在一个由 '0' 和 '1' 组成的二维矩阵内, 找到只包含 '1' 的最大正方形, 并返回其面积。

要求:

- 1) **状态定义**: 定义二维 DP 数组 ' $dp[i][j]$ ' 的确切物理含义。这个定义比较巧妙, 请准确描述。
- 2) **递推关系**: 写出状态转移方程。
- 3) **初值**: 说明 DP 数组的初始化。
- 4) **算法伪代码**: 给出求解该问题的伪代码。

问题 25: 地下城游戏 (Dungeon Game)

一个骑士被困在一个 ' $m \times n$ ' 的地下城中。地下城每个房间都可能有三种情况: 1. 非负整数, 表示房间里有魔力药水, 骑士的生命值会增加; 2. 负整数, 表示房间里有恶魔, 骑士的生命值会减少; 3. 0, 表示房间为空。骑士从左上角出发, 每次只能向右或向下移动, 最终到达右下角。骑士的生命值任何时候都必须大于 0。为了拯救公主, 骑士需要以尽可能高的初始生命值出发。请计算骑士需要的最低初始生命值。

要求:

- 1) **核心思想**: 说明为什么这个问题需要从终点向起点进行动态规划。
- 2) **状态定义**: 定义 DP 数组 ' $dp[i][j]$ ' 的含义: 从 ' (i, j) ' 出发到达终点所需的最少初始生命值。
- 3) **递推关系**: 写出状态转移方程。
- 4) **算法伪代码**: 给出求解该问题的伪代码。

问题 26: 旅行商问题 (TSP with Bitmask DP)

给定 ' n ' 个城市和一张完全图, 图中 ' $\text{dist}[i][j]$ ' 表示城市 ' i ' 到城市 ' j ' 的距离。寻找一条经过每个城市恰好一次并最终返回起点的最短路径 (哈密顿回路)。假设 ' n ' 的规模很小 (e.g., ' $n \leq 20$ ').

要求:

- 1) **状态定义:** 使用状态压缩 DP。定义 ' $\text{dp}[\text{mask}][i]$ ' 的含义, 其中 ' mask ' 是一个二进制位掩码。
- 2) **递推关系:** 写出状态转移方程。
- 3) **初值:** 说明 DP 数组的初始化。
- 4) **算法伪代码:** 给出求解该问题的伪代码。

问题 27: 找到最长的半回文子字符串

一个字符串是“半回文”的, 如果它可以通过不超过一次字符修改变为回文串。给定一个字符串 ' s ', 找到 ' s ' 中最长的半回文子字符串的长度。

要求:

- 1) **问题分解:** 将问题分解为两个子问题: 1. 判断任意子串 ' $s[i...j]$ ' 是否是半回文串。2. 寻找满足条件的最长子串。
- 2) **DP 设计:** 设计一个二维 DP ' $\text{is_palindrome}[i][j]$ ' 来预处理所有子串的回文性质, ' $\text{is_palindrome}[i][j]$ ' 可以存储从 ' i ' 到 ' j ' 的子串需要修改多少次才能变成回文。
- 3) **状态转移:** 写出 ' $\text{is_palindrome}[i][j]$ ' 的状态转移方程。
- 4) **最终求解:** 描述如何利用预处理的结果找到最长半回文子串。

问题 28: 数位 DP (Digit DP)

给定一个正整数 ' n ', 求 ' $[1, n]$ ' 区间内所有正整数中, 不包含"13" 并且本身是 13 的倍数的数字个数。

要求:

- 1) **核心思想:** 描述数位 DP 的基本思想, 即如何按位构造数字, 并处理“贴边” (limit) 和“前导零” (leading zero) 的情况。
- 2) **状态定义:** 设计一个记忆化搜索的函数签名或 DP 数组状态。状态应至少包含: 当前是第几位 ' pos '、前一位数字 ' prev_digit '、当前构造出的数字模 13 的余数 ' rem '、是否贴边 ' is_limit '。
- 3) **递推逻辑:** 描述在记忆化搜索函数内部的递推逻辑。
- 4) **问题求解:** 说明如何使用设计的函数来求解 ' $[1, n]$ ' 区间的问题。

问题 29：连接棒材的最低费用

你有 n 根棒材 `sticks`，其中 `sticks[i]` 是第 i 根棒材的长度。你需要将所有棒材连接成一根。连接两根长度为 x 和 y 的棒材的费用是 $x + y$ 。请找出将所有棒材连接成一根所需要的最低费用。

要求：这个问题可以用贪心法（哈夫曼编码思想）高效解决。但请你尝试用动态规划的思路来建模和分析。

- 1) **模型识别：**指出这个问题与哪个经典的区间 DP 问题相似。
- 2) **状态定义：**如果将棒材预先排序，定义 `dp[i][j]` 表示合并从 `sticks[i]` 到 `sticks[j]` 所有棒材的最低费用。
- 3) **递推关系：**写出状态转移方程。
- 4) **复杂度分析：**分析该 DP 解法的时间复杂度，并与贪心解法（使用优先队列）的复杂度进行比较。

问题 30：交错字符串 (Interleaving String)

给定三个字符串 `s1`、`s2`、`s3`，请你帮忙验证 `s3` 是否是由 `s1` 和 `s2` 交错组成的。两个字符串 `s` 和 `t` 的交错形式是指，将 `s` 和 `t` 分割成若干非空子串后， $s = s_1 + s_2 + \dots + s_n$ ， $t = t_1 + t_2 + \dots + t_m$ ，然后交替连接这些子串得到 $s_1 + t_1 + s_2 + t_2 + \dots$ 或 $t_1 + s_1 + t_2 + s_2 + \dots$ ，要求保持 `s` 和 `t` 内部的字符相对顺序不变。

要求：

- 1) **状态定义：**定义二维 DP 数组 `dp[i][j]` 的布尔含义。
- 2) **递推关系：**写出状态转移方程。
- 3) **初值：**说明 DP 数组的初始化。
- 4) **算法伪代码：**给出求解该问题的伪代码。

问题 31：完美平方数

给定一个正整数 n ，找到若干个完全平方数（比如 1, 4, 9, 16, ...）使得它们的和等于 n 。你需要让组成和的完全平方数的个数最少。

要求：

- 1) **模型识别：**指出该问题可以被看作哪种背包问题。
- 2) **状态定义与递推关系：**给出 DP 的状态定义和转移方程。
- 3) **算法伪代码：**给出求解该问题的伪代码。

问题 32：通配符匹配

给定一个字符串 `s` 和一个字符规律 `p`，实现一个支持 `“?”` 和 `“*”` 的通配符匹配。`“?”` 可以匹配任何单个字符。`“*”` 可以匹配任意字符串（包括空字符串）。

要求：

- 1) **状态定义**: 定义二维 DP 数组 `dp[i][j]` 的布尔含义。
- 2) **递推关系**: 写出状态转移方程。需要详细讨论当 `p[j-1]` 为普通字符、`'?'` 或 `'*'` 时的转移规则。
- 3) **初值**: 说明 DP 数组的初始化, 特别是 `dp[0][0]` 和第一行 `dp[0][j]` 的情况。
- 4) **算法伪代码**: 给出求解该问题的伪代码。

问题 33: 最大矩形

给定一个仅包含 `'0'` 和 `'1'` 的二维二进制矩阵, 找出只包含 `'1'` 的最大矩形, 并返回其面积。

要求:

- 1) **问题转化**: 描述如何将此问题转化为多个“柱状图中最大的矩形”问题。
- 2) **DP/辅助数组**: 定义一个辅助数组 (或 DP 数组), 例如 `height[j]`, 表示在当前行 `i`, 第 `j` 列向上连续的 `'1'` 的数量。
- 3) **核心子问题**: 简要描述如何解决“柱状图中最大的矩形”这个子问题 (通常使用单调栈)。
- 4) **算法伪代码**: 给出解决原始问题的整体算法伪代码。

问题 34: 形成目标字符串的方案数

给你两个字符串数组 `words` 和一个字符串 `target`。`words` 中所有字符串长度相同。你需要从 `words` 中选出一些列, 按顺序连接它们, 以形成 `target`。求形成 `target` 的方案数。由于答案可能很大, 请对 $10^9 + 7$ 取模。

要求:

- 1) **预处理**: 设计一个二维数组 `count[i][char]` 来预处理 `words` 数组, `count[i][char]` 表示在所有单词的第 `i` 列, 字符 `char` 出现的次数。
- 2) **状态定义**: 定义 `dp[i][j]` 表示使用 `words` 的前 `j` 列构成 `target` 的前 `i` 个字符的方案数。
- 3) **递推关系**: 写出状态转移方程。
- 4) **算法伪代码**: 给出求解该问题的伪代码。

问题 35: 学生出勤记录 II

可以用字符串表示一个学生的出勤记录, 其中的每个字符用来标记当天的出勤情况。记录中只含下面三种字符: `'A'`: 缺勤、`'L'`: 迟到、`'P'`: 到场。如果一个学生的出勤记录中不超过一个 `'A'` (缺勤) 并且不超过两个连续的 `'L'` (迟到), 那么这个学生会被授予奖励。给你一个整数 `n`, 表示出勤记录的长度, 请你返回所有可能的奖励记录的数量。答案可能很大, 需要对 $10^9 + 7$ 取模。

要求:

- 1) **状态定义**: 设计一个 DP 状态。由于限制条件涉及 `'A'` 的总数和 `'L'` 的连续数, 状态需要包含这些信息。例如, `dp[i][j][k]` 表示长度为 `i` 的字符串, 包含 `j` 个 `'A'`, 且以 `k` 个 `'L'` 结尾的方案数。
- 2) **递推关系**: 写出状态转移方程。

- 3) **初值**：说明 DP 数组的初始化。
- 4) **算法伪代码**：给出求解该问题的伪代码。

问题 36：扔鸡蛋问题 (超赞的鸡蛋掉落)

你面前有一栋 ' k ' 层的楼，你手中有 ' n ' 个一模一样的鸡蛋。你想知道存在哪一层 ' F ' ($0 \leq F \leq k$)，在这层楼及以下扔鸡蛋，鸡蛋不会碎；在这层楼及以上扔鸡蛋，鸡蛋会碎。在最坏情况下，你需要最少几次实验才能确定 ' F ' 的值？

要求：

- 1) **状态定义 (思路一)**：定义 ' $dp[i][j]$ ' 为有 ' i ' 个鸡蛋和 ' j ' 层楼时，最坏情况下需要的最少实验次数。写出其状态转移方程。
- 2) **状态定义 (思路二)**：改变视角，定义 ' $dp[i][j]$ ' 为有 ' i ' 个鸡蛋，进行 ' j ' 次实验，最多能确定的楼层数。写出其状态转移方程。
- 3) **比较分析**：比较两种 DP 思路的优劣和时间复杂度。
- 4) **算法伪代码**：给出基于思路二的算法伪代码。

问题 37：最长回文子序列

给定一个字符串 ' s '，找到其中最长的回文子序列，并返回该子序列的长度。子序列不要求连续。

要求：

- 1) **状态定义**：定义二维 DP 数组 ' $dp[i][j]$ ' 的确切物理含义。
- 2) **递推关系**：写出状态转移方程，需要考虑 ' $s[i] == s[j]$ ' 和 ' $s[i] != s[j]$ ' 两种情况。
- 3) **遍历顺序**：解释计算 DP 表时的正确遍历顺序，并说明原因。
- 4) **算法伪代码**：给出求解该问题的伪代码。

问题 38：不相交的线

在两条水平线上，分别有 ' $nums1$ ' 和 ' $nums2$ ' 数组中的数字。我们可以在 ' $nums1[i]$ ' 和 ' $nums2[j]$ ' 之间画一条线，当且仅当 ' $nums1[i] == nums2[j]$ ' 且我们画的线不与任何其他线相交（即使是在端点）。求最多可以画出多少条不相交的线。

要求：

- 1) **问题转化**：证明此问题等价于求解 ' $nums1$ ' 和 ' $nums2$ ' 的最长公共子序列问题。
- 2) **DP 求解**：基于上述转化，直接套用 LCS 的 DP 解法，写出状态定义、转移方程和伪代码。

问题 39: 贴纸拼词

我们有 n 种不同的贴纸。每种贴纸上都写着一个英语小写字母。给定一个字符串数组 `stickers`，其中 `stickers[i]` 代表第 i 种贴纸。你可以无限次地使用每种贴纸。目标是拼出给定的字符串 `target`，求最少需要多少张贴纸。如果无法拼出，返回 -1。

要求：

- 1) **状态定义：**这是一个对 `target` 的子序列进行拼凑的问题。可以使用状态压缩 DP，定义 `dp[mask]` 表示拼出 `target` 的子序列（由 `mask` 表示）所需的最少贴纸数。
- 2) **递推关系：**描述如何进行状态转移。对于每个状态 `mask`，可以尝试使用任意一张贴纸，看能将 `mask` 推进到哪个新状态。
- 3) **优化：**描述可能的优化，例如在状态转移时，总是尝试去填充 `mask` 中第一个为 0 的位对应的字符。
- 4) **算法伪代码：**给出求解该问题的伪代码。

问题 40: 最大化一张图中的路径价值

给你一个 $n \times n$ 的整数矩阵 `grid` 和一个整数 k 。你从 $(0, 0)$ 开始，最终要到达 $(n-1, n-1)$ 。你只能向下或向右移动。每次移动，路径的价值会加上你进入的单元格的值。此外，每当路径价值能被 k 整除时，该路径就是一个“有效路径”。返回所有从 $(0, 0)$ 到 $(n-1, n-1)$ 的路径中，有效路径的最大价值。如果不存在有效路径，返回 -1。

要求：

- 1) **状态定义：**传统的 `dp[i][j]` 只记录最大价值是不够的，因为它丢失了模 k 的信息。请设计一个包含余数信息的三维 DP 状态 `dp[i][j][rem]`。
- 2) **递推关系：**写出 `dp[i][j][rem]` 的状态转移方程。
- 3) **初值与结果：**说明 DP 数组的初始化，以及如何从最终的 DP 状态 `dp[n-1][n-1][0]` 获得答案。
- 4) **算法伪代码：**给出求解该问题的伪代码。