利用动态规划方法求解 LIS 问题. (以下"递增"指"严格递增")

nums 数组用于存储要求解的数列.

dp 数组: dp[i]存储以 nums[i]结尾的最长递增子序列的长度.

prev 数组: prev[i]存储 nums[i]的前驱节点索引, 用于回溯构建子序列.

maxLen: 记录最长递增子序列的长度.

allLIS 数组: 二维数组, 用于存储所有最长递增子序列.

currentLIS 数组: 用于在构建 allLIS 的递归中存储单个最长递增子序列.

maxIndices: 用于存储可回溯出一个最长递增子序列的所有索引

伪代码:

定义函数 findAllLIS(nums):

如果 nums 为空:

返回

令 n 为 nums 的长度

初始化 dp 数组, 长度为 n, 所有元素为 1

初始化 prev 数组, 长度为 n, 所有元素为 -1

初始化 maxLen 为 1

// 动态规划求解 LIS

对于 i 从 0 到 n-1:

对于 j 从 0 到 i-1:

如果 nums[i] > nums[j] 且 dp[i] < dp[j] + 1:

dp[i] = dp[j] + 1

prev[i] = j

如果 dp[i] > maxLen:
    maxLen = dp[i]

// 找到所有最长递增子序列的起始索引
初始化 maxIndices 为空列表
对于 i 从 0 到 n-1:
    如果 dp[i] == maxLen:
        将 i 添加到 maxIndices

// 回溯找到所有最长递增子序列
初始化 allLIS 为空列表
初始化 currentLIS 为空列表

定义函数 backtrack(index):
    如果 index == -1:
        将 currentLIS 添加到 allLIS
        调用 backtrack(prev[index])
        从 currentLIS 移除 nums[index]

对于 index 在 maxIndices 中的每个元素:
    调用 backtrack(index)

// 输出结果
打印 "所有最长的严格递增子序列是:"
对于 lis 在 allLIS 中的每个元素:

反转 LIS

打印 LIS 中的每个 num

给定或输入 nums

调用 findAllLIS(nums)

时间复杂度主要在双重循环处，为 $O(n^2)$.

举例：nums=[12, 1, 9, 6, 2, 8, 20, 17]

运行后

i	nums[i]	dp[i]	prev[i]
0	12	1	-1
1	1	1	-1
2	9	2	1
3	6	2	1
4	2	2	1
5	8	3	3
6	20	4	5
7	17	4	5

maxLen = 4 , maxIndices = [6, 7]

对于 index = 6，回溯得：[1, 6, 8, 20]

对于 index = 7，回溯得：[1, 6, 8, 17]

回溯这一块还有问题，没找全，prev[i] 可能有不只一种选择