【第二十七周】动态规划算法优化

72. 编辑距离

给你两个单词 `word1` 和 `word2`，请你计算出将 `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

示例 1：

```
输入：word1 = "horse", word2 = "ros"
输出：3
解释：
horse -> rorse (将 'h' 替换为 'r')
rorse -> rose (删除 'r')
rose -> ros (删除 'e')
```

309. 最佳买卖股票时机含冷冻期

给定一个整数数组，其中第 *i* 个元素代表了第 *i* 天的股票价格 。

设计一个算法计算出最大利润。在满足以下约束条件下，你可以尽可能地完成更多的交易（多次买卖一支股票）:

- 你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。
- 卖出股票后，你无法在第二天买入股票 (即冷冻期为 1 天)。

示例:

```
输入: [1,2,3,0,2]
输出:3
解释: 对应的交易状态为: [买入, 卖出, 冷冻期, 买入, 卖出]
```

1218. 最长定差子序列

给你一个整数数组 `arr` 和一个整数 `difference`，请你找出并返回 `arr` 中最长等差子序列的长度，该子序列中相邻元素之间的差等于 `difference` 。

子序列 是指在不改变其余元素顺序的情况下，通过删除一些元素或不删除任何元素而从 `arr` 派生出来的序列。

示例 1：

```
输入：arr = [1,2,3,4], difference = 1
输出：4
解释：最长的等差子序列是 [1,2,3,4]。
```

494. 目标和

给你一个整数数组 `nums` 和一个整数 `target` 。

向数组中的每个整数前添加 `'+'` 或 `'-'` ，然后串联起所有整数，可以构造一个 **表达式** ：

- 例如，`nums = [2, 1]` ，可以在 `2` 之前添加 `'+'` ，在 `1` 之前添加 `'-'` ，然后串联起来得到表达式 `"+2-1"` 。

返回可以通过上述方法构造的、运算结果等于 `target` 的不同 **表达式** 的数目。

示例 1：

```
输入：nums = [1,1,1,1,1], target = 3
输出：5
解释：一共有 5 种方法让最终目标和为 3 。
-1 + 1 + 1 + 1 + 1 = 3
+1 - 1 + 1 + 1 + 1 = 3
+1 + 1 - 1 + 1 + 1 = 3
+1 + 1 + 1 - 1 + 1 = 3
+1 + 1 + 1 + 1 - 1 = 3
```

91. 解码方法

一条包含字母 `A-Z` 的消息通过以下映射进行了 **编码** ：

```
'A' -> 1
'B' -> 2
...
'Z' -> 26
```

要 **解码** 已编码的消息，所有数字必须基于上述映射的方法，反向映射回字母（可能有多种方法）。例如，`"11106"` 可以映射为：

- `"AAJF"` ，将消息分组为 `(1 1 10 6)`

- `"KJF"` ，将消息分组为 `(11 10 6)`

注意，消息不能分组为 `(1 11 06)` ，因为 `"06"` 不能映射为 `"F"` ，这是由于 `"6"` 和 `"06"` 在映射中并不等价。

给你一个只含数字的 **非空** 字符串 `s` ，请计算并返回 **解码** 方法的 **总数** 。

题目数据保证答案肯定是一个 **32 位** 的整数。

示例 1：

```
输入：s = "12"
输出：2
解释：它可以解码为 "AB"（1 2）或者 "L"（12）。
```

486. 预测赢家

给定一个表示分数的非负整数数组。 玩家 1 从数组任意一端拿取一个分数，随后玩家 2 继续从剩余数组任意一端拿取分数，然后玩家 1 拿，…… 。每次一个玩家只能拿取一个分数，分数被拿取之后不再可取。直到没有剩余分数可取时游戏结束。最终获得分数总和最多的玩家获胜。

给定一个表示分数的数组，预测玩家1是否会成为赢家。你可以假设每个玩家的玩法都会使他的分数最大化。

示例 1：

```
输入：[1, 5, 2]
输出：False
解释：一开始，玩家1可以从1和2中进行选择。
如果他选择 2（或者 1 ），那么玩家 2 可以从 1（或者 2 ）和 5 中进行选择。如果玩家 2 选择了 5 ，那么玩家 1 则只剩下 1（或者 2 ）可选。
所以，玩家 1 的最终分数为 1 + 2 = 3，而玩家 2 为 5 。
因此，玩家 1 永远不会成为赢家，返回 False 。
```

1937. 扣分后的最大得分

给你一个 `m x n` 的整数矩阵 `points` （下标从 **0** 开始）。一开始你的得分为 `0` ，你想最大化从矩阵中得到的分数。

你的得分方式为：**每一行** 中选取一个格子，选中坐标为 `(r, c)` 的格子会给你的总得分 **增加** `points[r][c]` 。

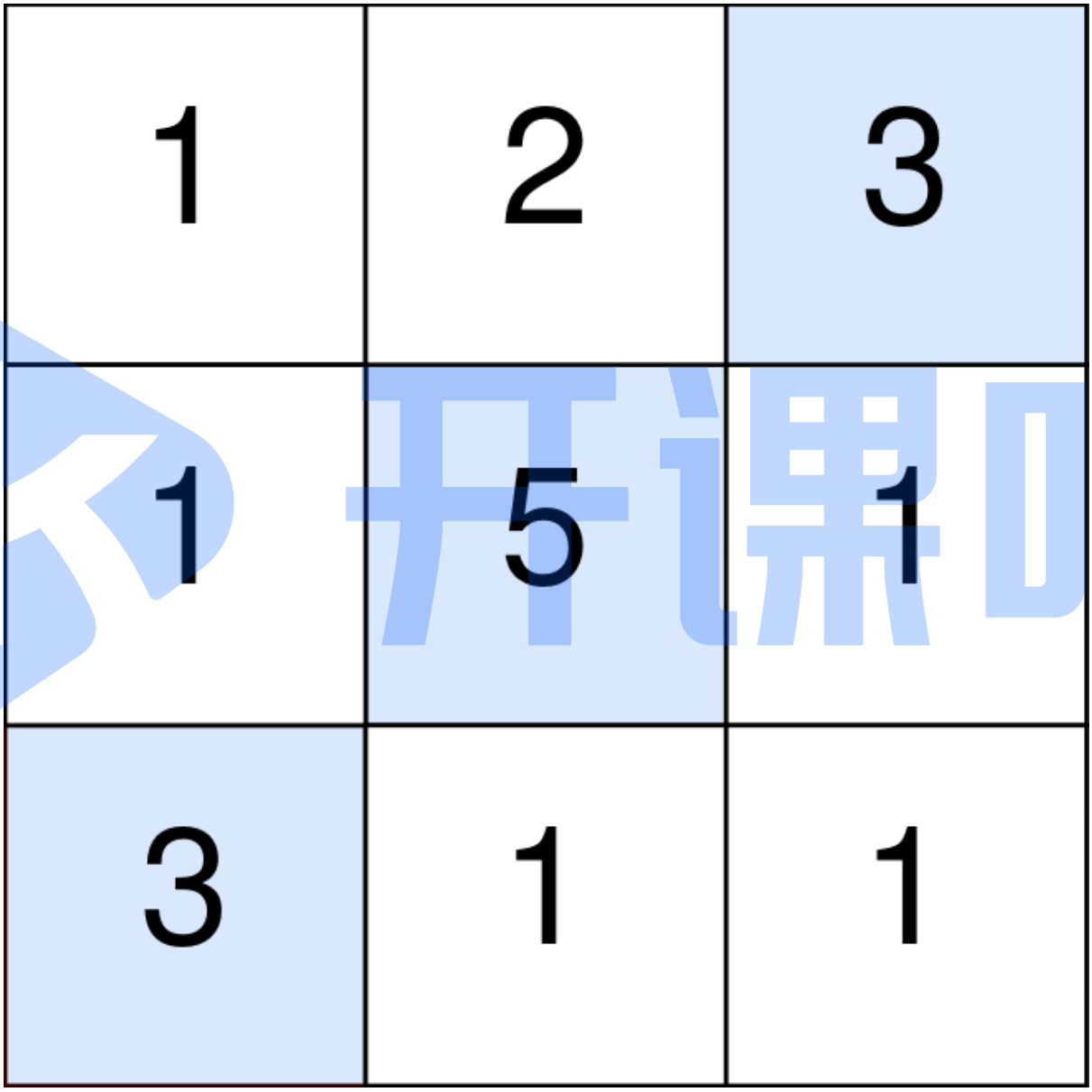
然而，相邻行之间被选中的格子如果隔得太远，你会失去一些得分。对于相邻行 `r` 和 `r + 1` （其中 `0 <= r < m - 1`），选中坐标为 `(r, c 1 )` 和 `(r + 1, c 2 )` 的格子，你的总得分 **减少** `abs(c 1 - c 2 )` 。

请你返回你能得到的 **最大** 得分。

`abs(x)` 定义为：

- 如果 `x >= 0` ，那么值为 `x` 。
- 如果 `x < 0` ，那么值为 `x` 。

示例 1：




```
输入：points = [[1,2,3],[1,5,1],[3,1,1]]
输出：9
解释：
蓝色格子是最优方案选中的格子，坐标分别为 (0, 2)，(1, 1) 和 (2, 0) 。
你的总得分增加 3 + 5 + 3 = 11 。
```

```
但是你的总得分需要扣除 abs(2 - 1) + abs(1 - 0) = 2 。
你的最终得分为 11 - 2 = 9 。
```

1312. 让字符串成为回文串的最少插入次数

给你一个字符串 `s` ，每一次操作你都可以在字符串的任意位置插入任意字符。

请你返回让 `s` 成为回文串的 **最少操作次数** 。

「回文串」是正读和反读都相同的字符串。

示例 1：

```
输入：s = "zzazz"
输出：0
解释：字符串 "zzazz" 已经是回文串了，所以不需要做任何插入操作。
```

714. 买卖股票的最佳时机含手续费

给定一个整数数组 `prices`，其中第 `i` 个元素代表了第 `i` 天的股票价格 ；整数 `fee` 代表了交易股票的手续费用。

你可以无限次地完成交易，但是你每笔交易都需要付手续费。如果你已经购买了一个股票，在卖出它之前你就不能再继续购买股票了。

返回获得利润的最大值。

注意：这里的一笔交易指买入持有并卖出股票的整个过程，每笔交易你只需要为支付一次手续费。

示例 1：

```
输入：prices = [1, 3, 2, 8, 4, 9], fee = 2
输出：8
解释：能够达到的最大利润:
在此处买入 prices[0] = 1
在此处卖出 prices[3] = 8
在此处买入 prices[4] = 4
在此处卖出 prices[5] = 9
总利润: ((8 - 1) - 2) + ((9 - 4) - 2) = 8
```

887. 鸡蛋掉落

给你 `k` 枚相同的鸡蛋，并可以使用一栋从第 `1` 层到第 `n` 层共有 `n` 层楼的建筑。

已知存在楼层 `f` ，满足 `0 <= f <= n` ，任何从 **高于** `f` 的楼层落下的鸡蛋都会碎，从 `f` 楼层或比它低的楼层落下的鸡蛋都不会破。

每次操作，你可以取一枚没有碎的鸡蛋并把它从任一楼层 `x` 扔下（满足 `1 <= x <= n`）。如果鸡蛋碎了，你就不能再次使用它。如果某枚鸡蛋扔下后没有摔碎，则可以在之后的操作中 **重复使用** 这枚鸡蛋。

请你计算并返回要确定 `f` **确切的值** 的 **最小操作次数** 是多少？

示例 1：

```
输入：k = 1, n = 2
输出：2
解释：
鸡蛋从 1 楼掉落。如果它碎了，肯定能得出 f = 0 。
否则，鸡蛋从 2 楼掉落。如果它碎了，肯定能得出 f = 1 。
如果它没碎，那么肯定能得出 f = 2 。
因此，在最坏的情况下我们需要移动 2 次以确定 f 是多少。
```