LeetCode 题库分门别类详细解析笔记（第9章）

第9章 动态规划基础（21题） 3

9-2 第一个动态规划问题 Climbing Stairs（3题） 3

 例题1：LeetCode 第 70 题 3

 练习1：LeetCode 第 120 题 Triangle 3

 练习2：LeetCode 第 64 题 Minimum Path Sum 3

9-3 第 2 个动态规划问题——整数分割问题（5题） 3

 例题1：LeetCode 第 343 题 3

 练习1：LeetCode 第 279 题 3

 练习2：LeetCode 第 91 题 3

 练习3：LeetCode 第 62 题 3

 练习4：LeetCode 第 63 题 3

9-4 状态和状态转移（4题） 3

 例题1：LeetCode 第 198 题 3

 练习1：LeetCode 第 213 题 House Robber II 4

 练习2：LeetCode 第 337 题 House Robber III 4

 练习3：LeetCode 第 309 题 Best Time to Buy and Sell Stock with Cooldown 4

9-7 面试中的 0-1 问题（6题） 4

 例题1：LeetCode 第 416 题 Partition Equal Subset Sum 4

 练习1：LeetCode 第 322 题 Coin Change 4

 练习2：LeetCode 第 377 题 Combination Sum IV 4

 练习3：LeetCode 第 474 题 Ones and Zeroes 4

 练习4：LeetCode 第 139 题 Word Break 4

 练习5：LeetCode 第 494 题 Target Sum 4

9-8 最长上升子序列问题（2题） 4

 例题1：LeetCode 第 300 题 Longest Increasing Subsequence 4

9-9 LCS，最短路、求动态规划的具体解以及更多（1题） 4

 练习1：LeetCode 第 300 题 Longest Increasing Subsequence 5

模板：

题目要求：

题目难度：

中文网址：

英文网址：

求解关键：

我的解答：

参考资料：

思考总结：

第9章 动态规划基础（21题）

很多同学听到“动态规划”的名称可能会望而生畏，觉得动态规划的问题都很复杂。但其实，动态规划本质依然是递归算法，只不过是满足特定条件的递归算法。在这一章里，我们就来逐步解开动态规划的神秘面纱。

9-1 从斐波拉契数列的实现谈起

1、按照斐波拉契数列的定义，我们最初的实现虽然是没有问题的，但是真正运行起来，我们会发现效率非常低，这是因为我们将很多规模较小的问题进行了重复计算：

为了计算 f(5)，我们要计算 f(4) 与 f(3)，

为了计算 f(4)，我们要计算 f(3) 与 f(2)...

从这两行我们就可以看出 f(3) 被计算了两次。原因是因为：我们的实现与我们的思考不一致，请大家仔细想一想，**如果不让你用计算机，让你使用纸笔计算，你会怎么做？**。

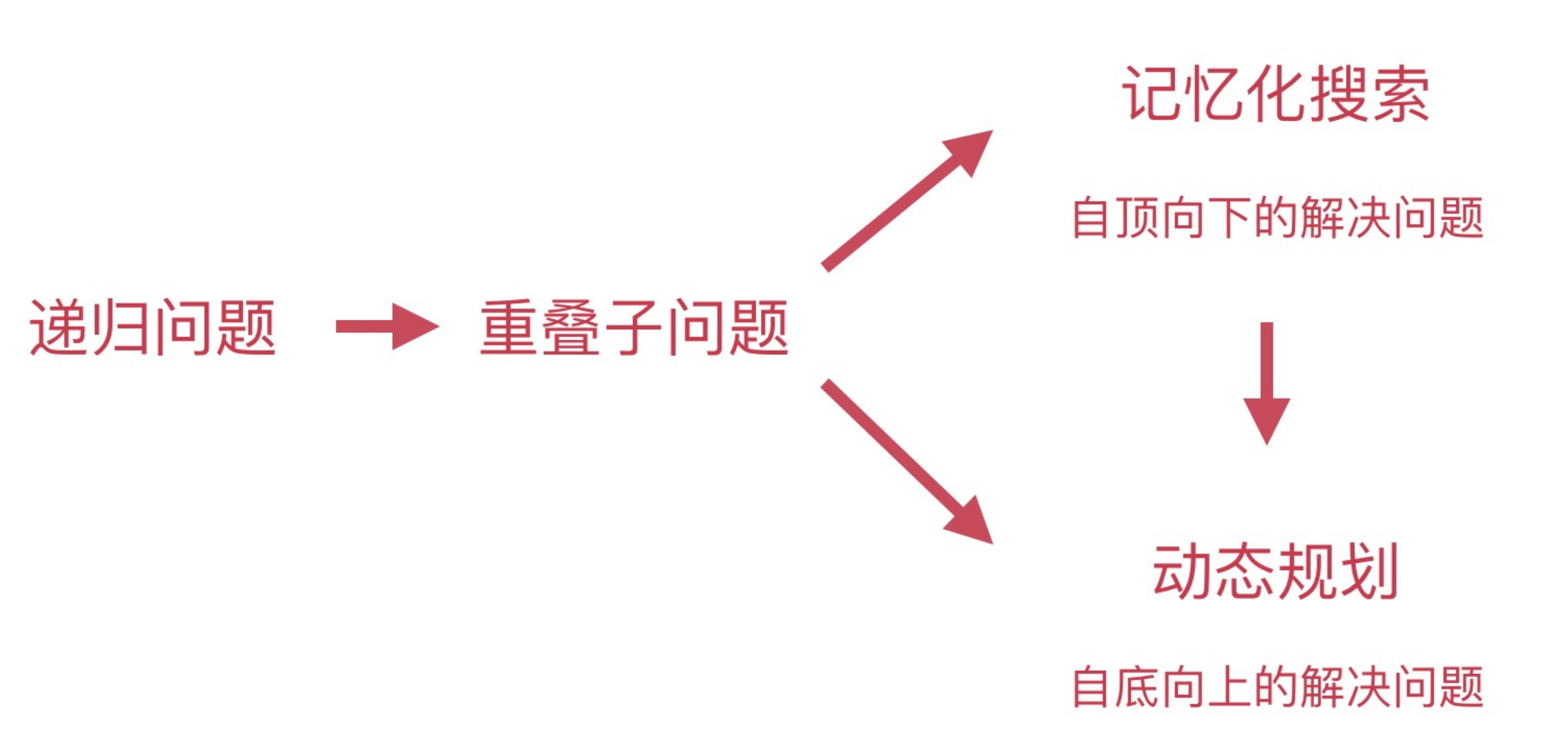
2、解决办法：很简单，既然我们进行了重复计算，我们就设置缓存就可以了，这一步叫“**记忆化搜索**”，“记忆化搜索”是**自顶向下**思考问题的时候，如果我们发现有大量重叠子问题，就一定要加上缓存。

3、既然定义是**自顶向下**的，我们不妨**自底向上**来实现，很简单，我们使用循环就可以完成。

4、什么是动态规划？（谷歌搜索“动态规划”可以获得很多信息。）

（来自维基百科）动态规划（英语：Dynamic programming，简称DP）是一种在数学、管理科学、计算机科学、经济学和生物信息学中使用的，通过把原问题分解为相对简单的子问题的方式求解复杂问题的方法。

5、我们通常使用“记忆化搜索”来帮助思考这个问题，但是使用“动态规划”来实现。



6、如果我们的动态规划应用于最优化问题，那么这样的动态规划问题叫做具有**最优子结构**。

本节练习：

（1）使用不同的方法实现斐波拉契数列数列；记忆化搜索和动态规划。（2）廖雪峰网站上的斐波拉契数列实现。

9-2 第一个动态规划问题 Climbing Stairs（3题）

1、首先来看下面的这个问题。

* 例题1：LeetCode 第 70 题 Climbing Stairs

题目要求：爬楼梯。假设你正在爬楼梯。需要 n 步你才能到达楼顶。每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？

题目难度：**简单**。

中文网址：https://leetcode-cn.com/problems/climbing-stairs/description/

英文网址：https://leetcode.com/problems/climbing-stairs/description/

求解关键：分析这道问题我们还是使用数学的方法，它们分别是我们非常熟悉的加法计数原理和乘法计数原理。分析如下：

要爬到第 n 层楼梯，可以有两种途径，由于这两种途径都可以达到目的，所以我们可以分别计算这两种途径分别可以使用多少种不同的方法爬到楼顶，根据**加法计数原理**，应该把它们**相加**，这一步用到加法计数原理，因为它们都可以完成我们要做的那件事情。

第 1 种途径：分两步：第 1 步，先爬到第 n-1 层，再爬 1层。

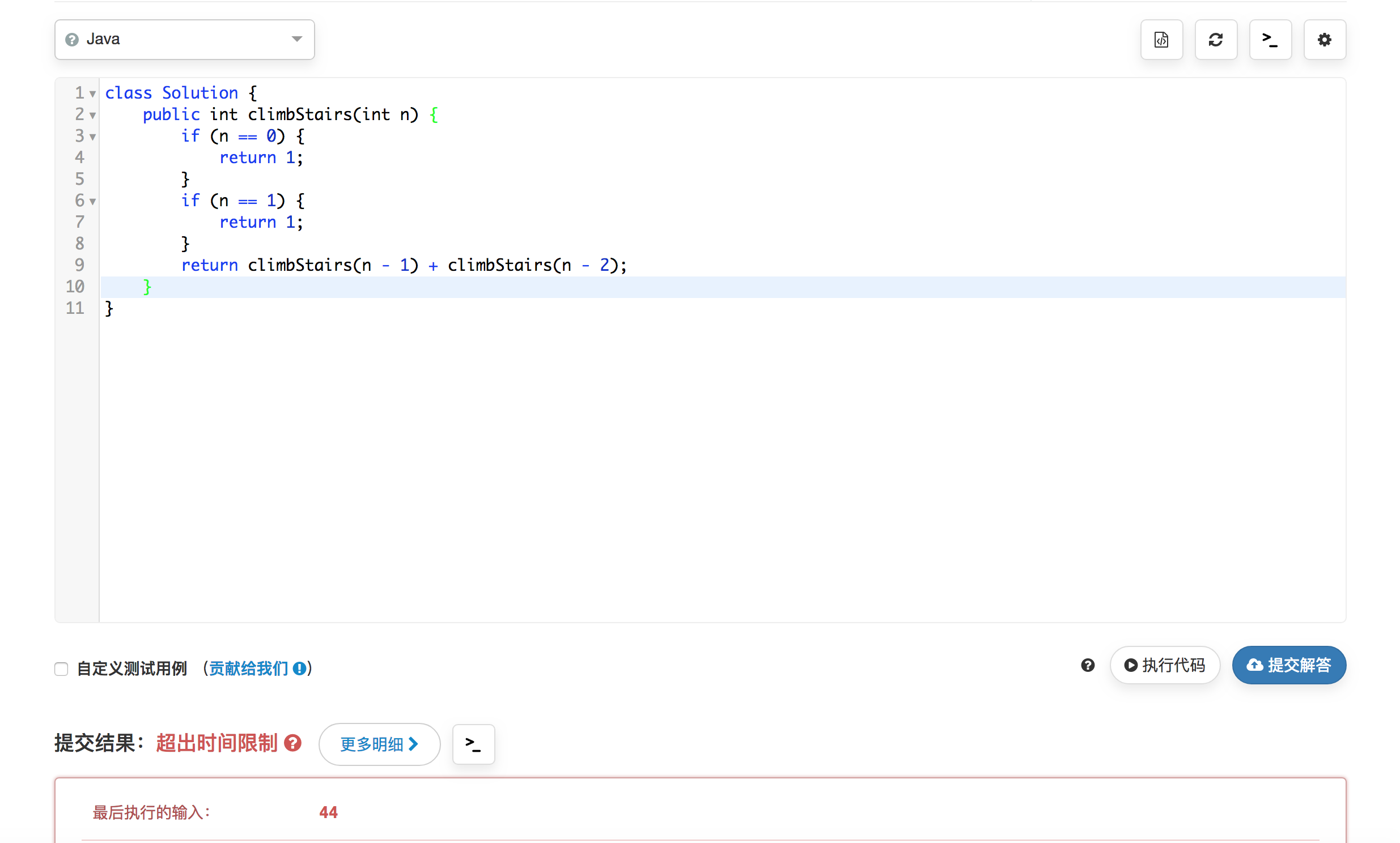
爬到第 n-1 层，有 f(n-1) 种方法，再爬 1 层，只有 1 种方法，因此这种途径一共有 f(n-1)\*1 种方法，这一步用到了**乘法计数原理**，每一步都只是完成这件事情的一部分，它们合起来完成了这件事情；

第 2 种途径：分两步：第 1 步，先爬到第 n-2 层，再爬 2层。

爬到第 n-1 层，有 f(n-2) 种方法，再爬 2 层，只有 1 种方法，因此这种途径一共有 f(n-2)\*1 种方法；

因此，要爬到第 n 层楼梯，一共有 f(n-1)\*1 + f(n-2)\*1 种方法。于是我们得到递推关系式：f(n) = f(n-1) + f(n-2)。

（呈现树形结构。）



我的解答：https://gist.github.com/liweiwei1419/580c3ac7bc1af85041d76e55103e9013

参考资料：

思考总结：这个爬楼梯的问题，**使用了与斐波拉契数列不一样的问题背景**，但通过我们的分析，从得到的递推关系式上，这道问题的本质就是一个斐波拉契数列。所以我们总结一下，我们思考这个问题的路径：（1）寻找递推关系式（2）发现重叠子问题（3）使用记忆化搜索（加上缓存）（4）使用动态规划

2、下面我们来完成几个 LeetCode 上的问题熟悉动态规划的应用，重点理解以下几点：（1）

我们是如何将原问题分解成更小的子问题，然后我们假设这些子问题都处理好了，那么这些子问题的解就组成了我们原问题的解。

* 练习1：LeetCode 第 120 题 Triangle

题目要求：三角形最小路径和。给定一个三角形，找出自顶向下的最小路径和。每一步只能移动到下一行中相邻的结点上。

题目难度：**中等**。

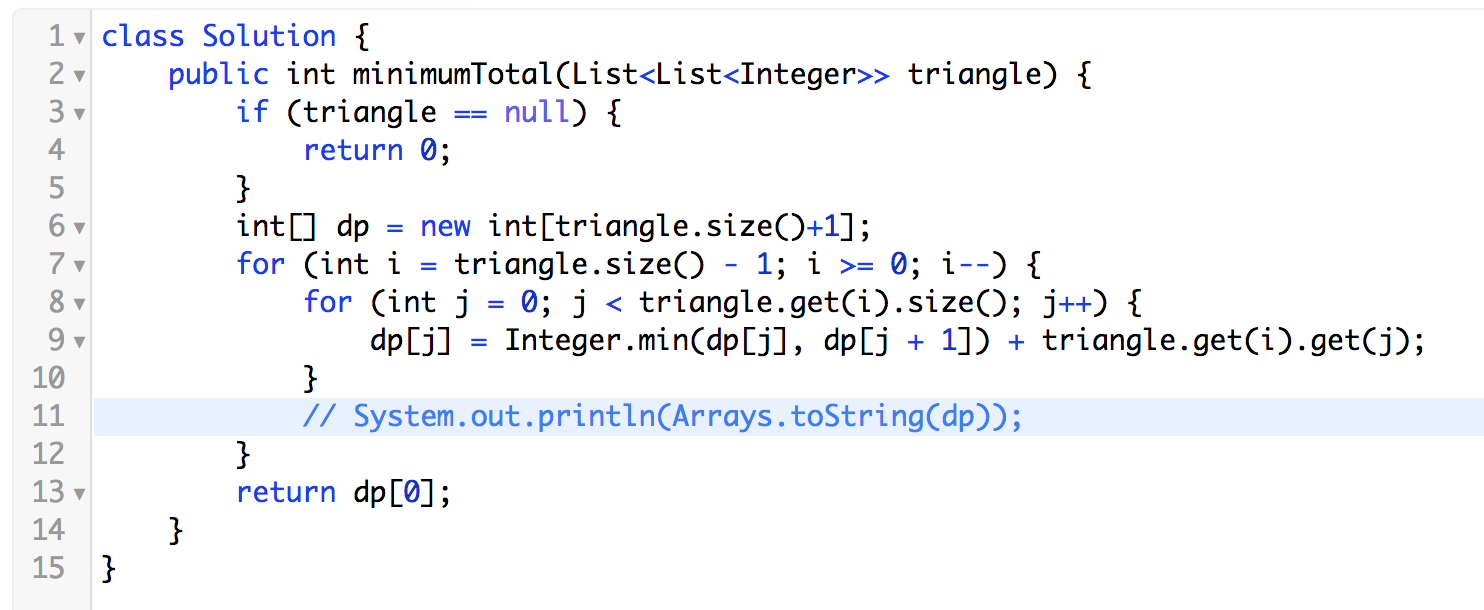
中文网址：https://leetcode-cn.com/problems/triangle/description/

英文网址：https://leetcode.com/problems/triangle/description/

求解关键：使用动态规划解决。

注意：（1）相邻问题；（2）最后一层元素的个数也就是这个图形的高度。（3）自顶向下走，每一步不能选最小值，这是因为我们每一步是有限制的，只能走左孩子或者右孩子，但是自底向上走，就能够找到这个最短路径值。

我的解答：



参考资料：

思考总结：思路比较奇特，并且还要注意很多边界问题，建议重刷。

（1）自顶向下考虑问题，使用的是记忆化搜索（我的 Python 解法）；

（2）自底向上考虑问题，使用的就是动态规划了（我的 Java 解法）。

* 练习2：LeetCode 第 64 题 Minimum Path Sum

题目要求：最小路径和。给定一个包含非负整数的 m x n 网格，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。

题目难度：**中等**。

中文网址：https://leetcode-cn.com/problems/minimum-path-sum/description/

英文网址：https://leetcode.com/problems/minimum-path-sum/description/

求解关键：

（1）不用开额外的空间，直接在 给出的 grid 上修改。

（2）那如果我不让你在原来的 grid 上修改，我们也不必创建和 grid 同规模的二维数组，完全可以用一维数组就可以完成任务。

（3）继续（2），有没有可能使用常数的空间复杂度就解决这个问题呢？

我的解答：

参考资料：

思考总结：很常规的一个问题。难度不应该设置为中等。

刘宇波老师给出的思考题：（1）非负整数的限定是为什么？（2）只能从左上到右下的限定又是为了什么？

9-3 第 2 个动态规划问题——从整数分割问题理解最优子结构（5题）

1、首先我们来看 LeetCode 第 343 题， 其实动态规划也包含了暴力求解，只不过我们按照一定规律，并且是在假设规模更小的问题已经得到解决的情况下，得到了我们原先要解决的那个规模的问题的解，我个人认为技巧在于“分类讨论”，而“分类讨论”的关键就在于“不重不漏”。

* 例题1：LeetCode 第 343 题 Integer Break

题目要求：整数拆分。给定一个正整数 n，将其拆分为至少两个正整数的和，并使这些整数的乘积最大化。 返回你可以获得的最大乘积。例如，给定 n = 2，返回1（2 = 1 + 1）；给定 n = 10，返回36（10 = 3 + 3 + 4）。注意：你可以假设 n 不小于2且不大于58。

题目难度：**中等**。

中文网址：https://leetcode-cn.com/problems/integer-break/description/

英文网址：https://leetcode.com/problems/integer-break/description/

求解关键：这道题解题的关键在于“至少分割成两个正整数”，从这个角度出发，就能够得到我们“自顶向下”思考这个问题的路径，进而使用“记忆化搜索”或者“动态规划”得到原问题的解。

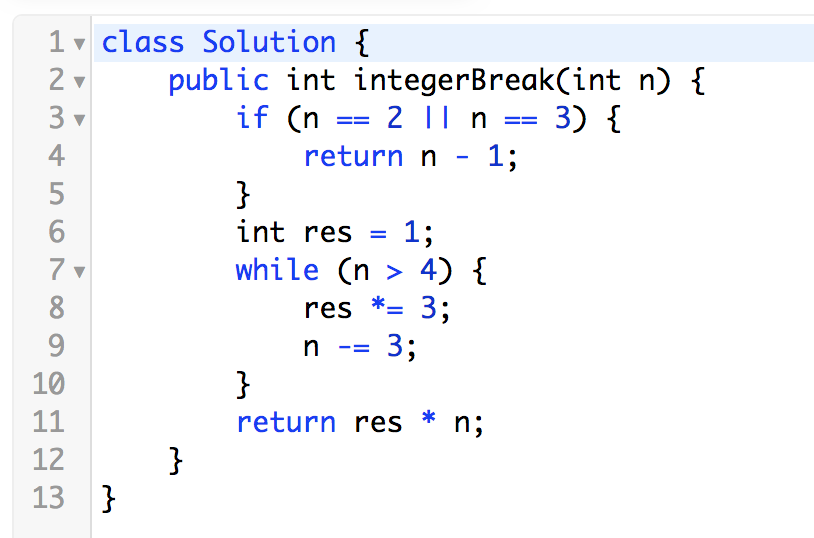
思路1：记忆化搜索（递归 + 缓存）。

思路2：动态规划（在想清楚了自顶向下的结构以后，写出动态规划的解是十分自然的，可以多写几次体会一下）。

思路3：找规律。这个规律要写到 10 左右才能看清楚。

|  |  |  |
| --- | --- | --- |
| 1 | 不能拆分 | 1 |
| 2 | 不能拆分 | 1 |
| 3 | 2 + 1 | 2 |
| 4 | 2 + 2 | 4 |
| 5 | 3 + 2 | 6 |
| 6 | 3 + 3 | 9 |
| 7 | 3 + 4 | 12 |
| 8 | 3 + 3 + 2 | 18 |
| 9 | 3 + 3 + 3 | 27 |
| 10 | 3 + 3 + 4 | 36 |
| 11 | 3 + 3 + 3 + 2 | 54 |
| 12 | 3 + 3 + 3 + 3 | 81 |

发现了规律以后，代码写起来，就很简洁了，当然比起前面并不好理解。



我的解答：

参考资料：

思考总结：对于这道问题，我们还是先画出树形结构，然后发现有大量重叠子问题，因此可以“记忆化搜索”，还可以“动态规划”。

2、什么是“最优子结构”？

我们通过求解子问题得到的最优解，组成了我们规模更大的原问题的最优解，这样的动态规划问题，我们称之为具有“最优子结构”。

3、动态规划问题通常应用的场景是：我们直接求解这个问题感觉难度较大，但是我们把这个问题拆分为规模更小的问题的时候，这个问题的解通常也就能够找到，这样的解决问题的实现通常都要借助递归来实现。

下面完成一些练习，重点体会什么是**“最优子结构”**。

* 练习1：LeetCode 第 279 题

题目要求：**求一个正整数能分解成完全平方数之和的最少个数。**给定正整数 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。

题目难度：**中等**。

中文网址：https://leetcode-cn.com/problems/perfect-squares/description/

英文网址：https://leetcode.com/problems/perfect-squares/description/

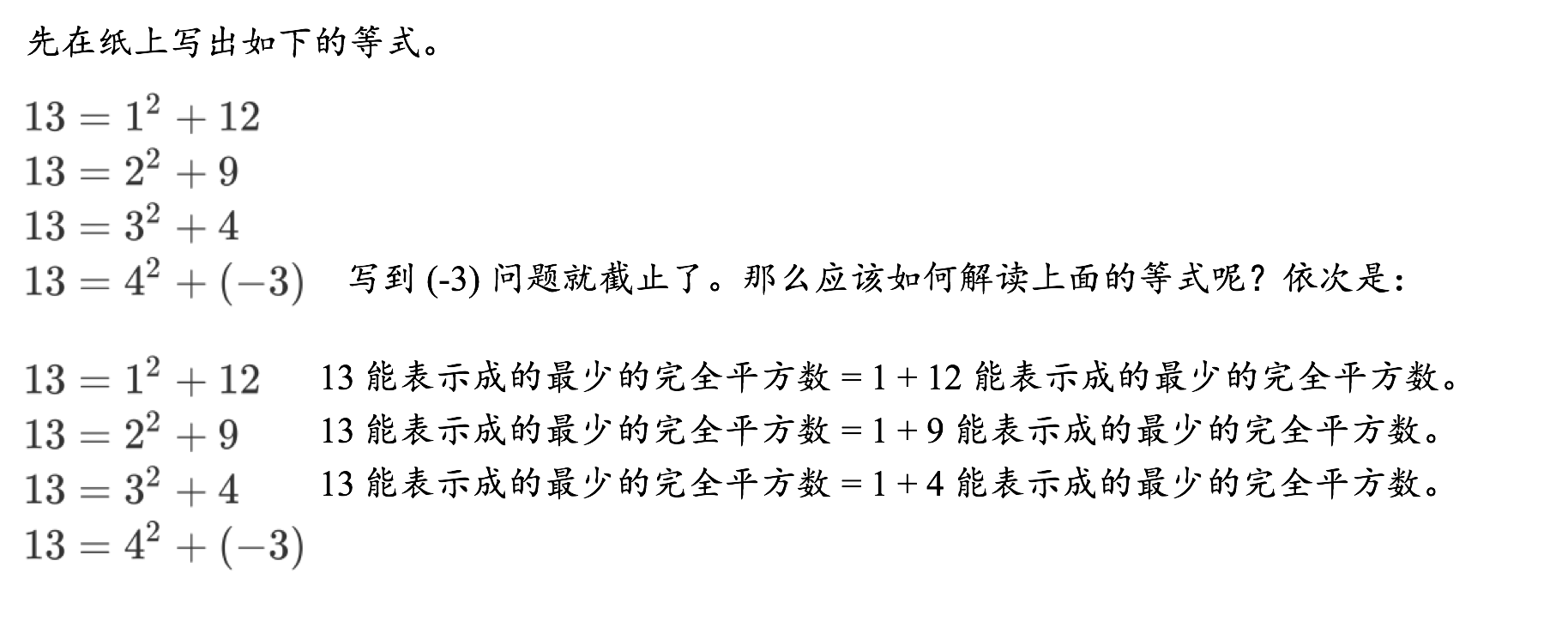
求解关键：

1、可以在图论中建模；

2、发现规律以后，不论是使用记忆化搜索，还是动态规划，都不难。

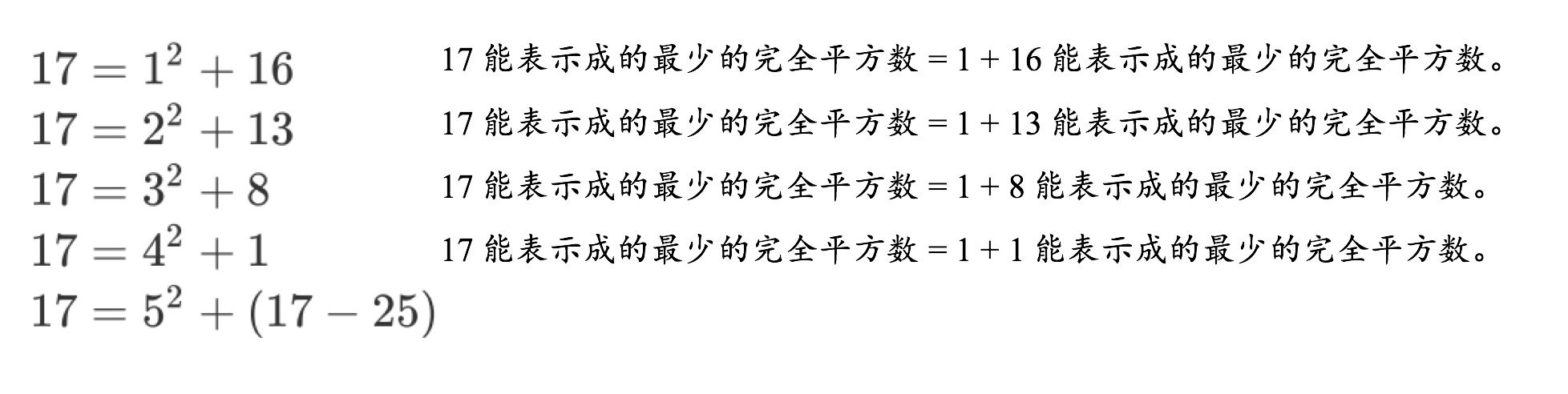
例如：

如果自己是完全平方数，就返回1。否则就是如下所有情况的最小值，我们以 13 为例进行说明：



13 得到的解为 2，其实就是 第 2 行和第 3 行的情况。

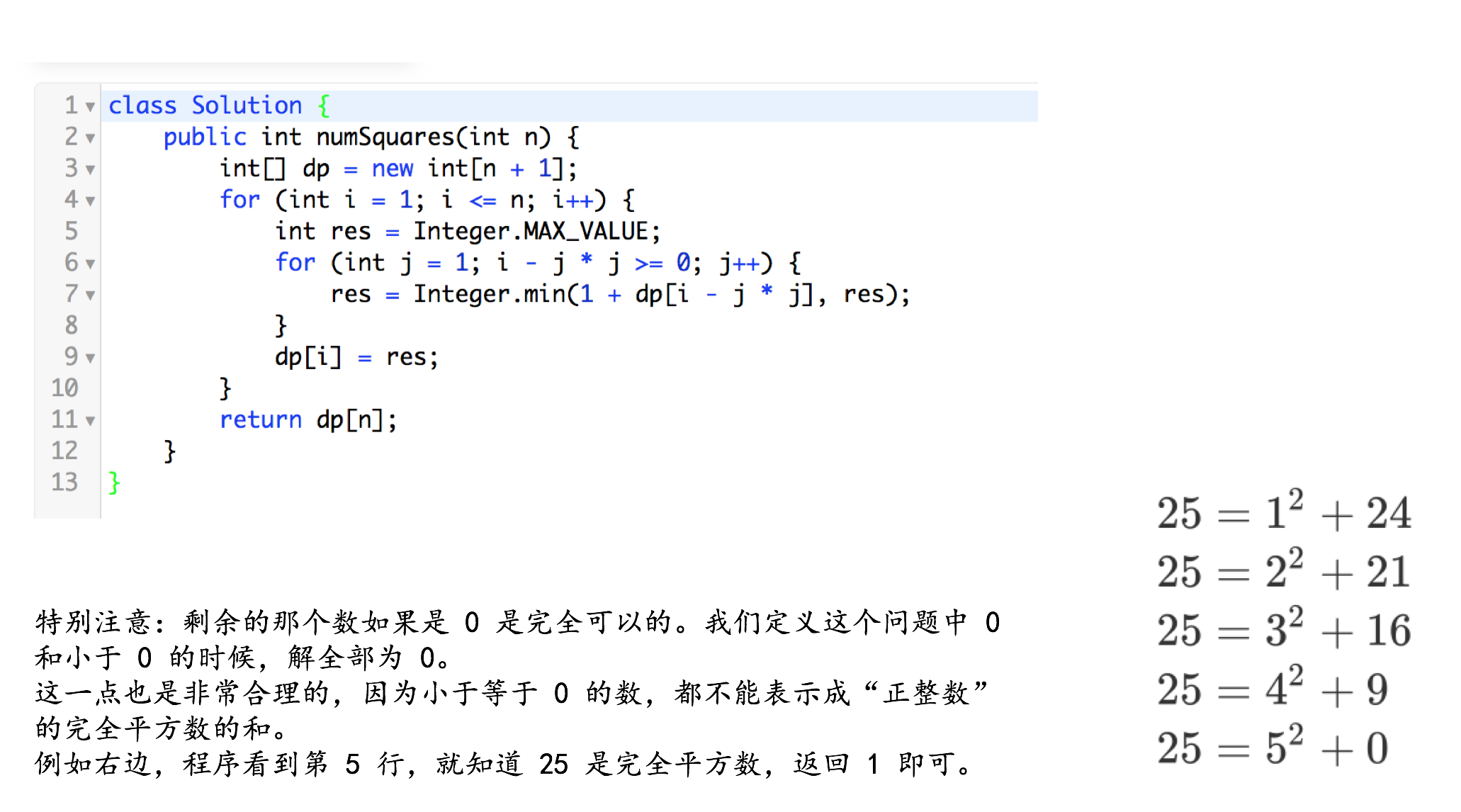
再以 17 为例：



17 得到的解也为2，看第 1 行就知道了。

特别注意：剩余的那个数如果等于 0 是完全可以的。我们定义这个问题中 0 和小于 0 的时候，解全部为 0。这一点也是非常合理的，因为小于等于 0 的数，都不能表示成“正整数”的完全平方数的和。此时当前考虑的这个数，就一定是完全平方数，直接返回 1 就可以了。例如：

我的解答：



代码实现要留意的地方：（1）因为大的值要依赖小的值，所以求解 25 会依赖比它小的值，这是设立外层循环的原因；（2）内层循环的终止条件是 i-j\*j>=0，体会这里 = 0 是为什么。（3）既然是求最小值，默认值就应该是一个很大的值。

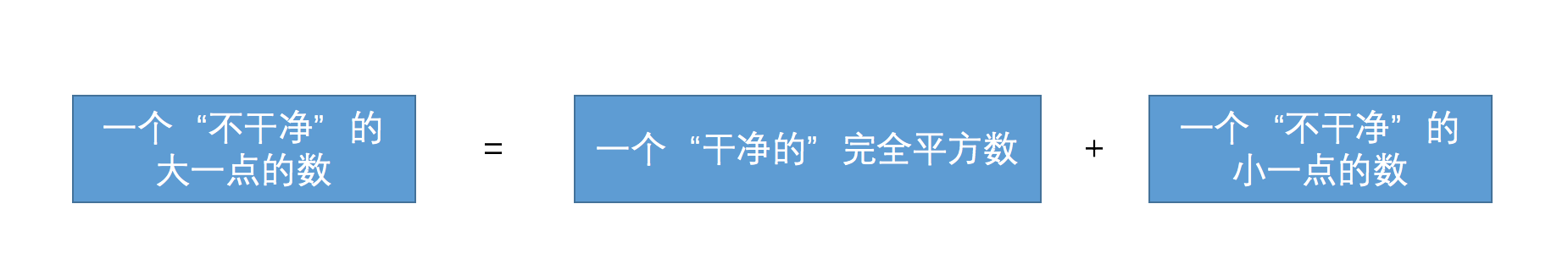
参考资料：

1、尝试使用广度优先遍历来解决一下。

思考总结：有很多解法。建议重刷。

维基百科中关于“动态规划”的相关叙述：动态规划只能应用于有最优子结构的问题。最优子结构的意思是局部最优解能决定全局最优解（对有些问题这个要求并不能完全满足，故有时需要引入一定的近似）。简单地说，问题能够分解成子问题来解决。

个人总结：这个问题的关键就在于“拆”，既然可以“拆”成多个的情况，那么最基本的情况就是“拆”成两个，这两个中，有一个是“干净”的完全平方数，还有一个是没有被“拆”干净的数（对于小的数我们人可以一眼看出，计算机看不出），所以还要继续“拆”。所以递归结构是这样的：



* 练习2：LeetCode 第 91 题 Decode Ways

题目要求：求解数字字符串的解码方法的总数。给定一个只包含数字的非空字符串，请计算解码方法的总数。

题目难度：**中等**。

中文网址：https://leetcode-cn.com/problems/decode-ways/description/

英文网址：https://leetcode.com/problems/decode-ways/description/

求解关键：

例：12321。

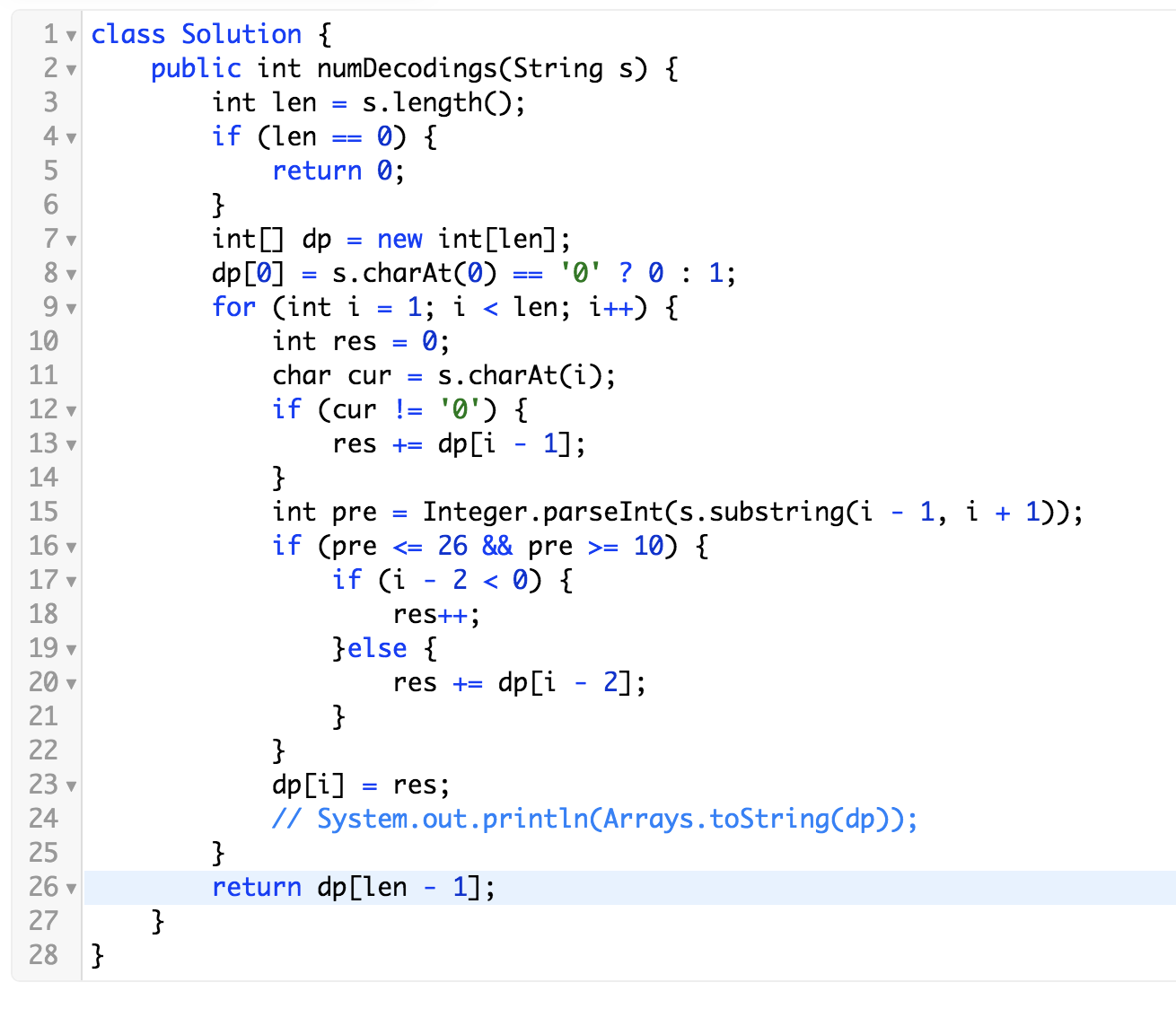
假设我们已经解决了 dp[0] 和 dp[1] ，从 dp[2] 开始考虑。

考虑num[2]：（1）如果 num[2] 不等于0，那么 dp[2] 的情况和 dp[1] 是一样的，完成编码，这是一种情况；

（2）如果 num[2] 跟前面的 num[1] 合起来能够组成一个字母，那么 dp[2] 和 dp[0] 是一样的，完成编码，这是一种情况。

两种情况都能完成编码，求总数，其实就是他们的和。这里也有加法计数原理的应用。

我的解答：



参考资料：还要一种解法，视频网站中的。

思考总结：**非常重要的一道题，建议重刷，2种解法，同等重要**。

重点考虑一下数组下标越界的时候，其实很简单，在原来基础上 + 1 就可以了。例如：12。2 的索引是 1 ，1-2 <-1 ，但是 12 又可以编码，所以是一种方案。

* 练习3：LeetCode 第 62 题 Unique Paths

题目要求：不同路径。一个机器人位于一个 m x n 网格的左上角 （起始点在下图中标记为“Start” ）。机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为“Finish”）。问总共有多少条不同的路径？

题目难度：**中等**。

中文网址：https://leetcode-cn.com/problems/unique-paths/description/

英文网址：https://leetcode.com/problems/unique-paths/description/

求解关键：

我的解答：

参考资料：

思考总结：

* 练习4：LeetCode 第 63 题 Unique Paths II

题目要求：不同路径 II。一个机器人位于一个 m x n 网格的左上角 （起始点在下图中标记为“Start” ）。机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为“Finish”）。现在考虑网格中有障碍物。那么从左上角到右下角将会有多少条不同的路径？网格中的障碍物和空位置分别用 1 和 0 来表示。说明：m 和 n 的值均不超过 100。

题目难度：**中等**。

中文网址：https://leetcode-cn.com/problems/unique-paths-ii/description/

英文网址：https://leetcode.com/problems/unique-paths-ii/description/

求解关键：

我的解答：

参考资料：

思考总结：

9-4 状态和状态转移（4题）

1、本节我们要聊一聊什么是“状态”以及什么是“状态转移”，它们是非常重要的概念，其实我们之前解决的问题，不妨也可以用“状态”以及“状态转移方程”的概念去思考它们。

* 例题1：LeetCode 第 198 题

题目要求：打家劫舍。你是一个专业的小偷，计划偷窃沿街的房屋。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。给定一个代表每个房屋存放金额的非负整数数组，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。

题目难度：简单。

中文网址：https://leetcode-cn.com/problems/house-robber/description/

英文网址：https://leetcode.com/problems/house-robber/description/

求解关键：这是一个最优化问题，它的解来自于解空间的所有组合里面的那个最优解，所以我们说，解空间是一个组合的解空间。

**定义“状态”，思考“转态转移方程”**。

“状态”：“考虑偷取从第 0 号开始的所有房屋，偷窃得到的最高金额。” 注意我们定义的“状态”中的关键字：“考虑”，“从第 几 号开始”，“所有”，“最高金额”，这些关键字对于我们的“状态转移方程”或者称之为“状态转移等式”是十分重要的。

为了书写“状态转移方程”方便，我们设置一个函数，函数的自变量为 i，则 f(i) 表示：“考虑偷取从第 i 号开始的所有房屋，偷窃得到的最高金额。”

“状态转移方程”： f(0) 可能来自于以下所有的值：

v(0) + f(2) 、v(1) + f(3) 、v(2) + f(4) 、...、v(n-3) + f(n-1)、v(n-1) 。从它们之中找到最小值即可。我们再回顾一下“动态规划”的定义，我们是不是在求解这个问题的过程中，把一个规模较大的问题转化成了一些“规模较小”的问题的“函数”（对于这道题而言，就是求它们的最小值）。我们再体会一下，“记忆化搜索”和递归是不是密不可分。

以上是“记忆化搜索”的实现，下面我们来思考“动态规划”是如何实现的？（要从后往前去填写 dp[]）。

我的解答：

参考资料：

思考总结：1、“状态”定义了我们要解决的问题是什么。（2）“状态转移方程”或“状态转移等式”则告诉了我们如何解决这个问题，还是那句话，解决这个问题，其实就是解决比原问题更小规模的问题，然后综合考虑原问题的解，这个综合考虑要根据问题的情境来，有的时候是求最小值，有的时候是求最大值，还有可能更复杂。

2、如果我们改变对“状态”的定义，那么“状态转移方程”就相应会有变化，只要在它们都是正确的前提下，都是可以得到正确的解。

* 练习1：LeetCode 第 213 题 House Robber II

题目要求：打家劫舍 II。你是一个专业的小偷，计划偷窃沿街的房屋，每间房内都藏有一定的现金。这个地方所有的房屋都围成一圈，这意味着第一个房屋和最后一个房屋是紧挨着的。同时，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。给定一个代表每个房屋存放金额的非负整数数组，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。

题目难度：中等。

中文网址：https://leetcode-cn.com/problems/house-robber-ii/description/

英文网址：https://leetcode.com/problems/house-robber-ii/description/

求解关键：

我的解答：

参考资料：

思考总结：

* 练习2：LeetCode 第 337 题 House Robber III

题目要求：打家劫舍 III。小偷又发现一个新的可行窃的地点。 这个地区只有一个入口，称为“根”。 除了根部之外，每栋房子有且只有一个父房子。 一番侦察之后，聪明的小偷意识到“这个地方的所有房屋形成了一棵二叉树”。 如果两个直接相连的房子在同一天晚上被打劫，房屋将自动报警。在不触动警报的情况下，计算小偷一晚能盗取的最高金额。

题目难度：中等。

中文网址：https://leetcode-cn.com/problems/house-robber-iii/description/

英文网址：https://leetcode.com/problems/house-robber-iii/description/

求解关键：

我的解答：

参考资料：

思考总结：

* 练习3：LeetCode 第 309 题 Best Time to Buy and Sell Stock with Cooldown

题目要求：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)。

题目难度：中等。

中文网址：

https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/description/

英文网址：

https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/description/

求解关键：

我的解答：

参考资料：

思考总结：

9-5 0-1 背包问题—— Knapsack01（6题）

1、问题背景：给了指定的物品和对应的重量和价值，计算在限定总重量的时候，物品组合的最大总价值。这里说 0-1 背包问题是很有计算机语言味道的，因为在计算机中 0 和 1 就代表是否，因此 0-1 的意思就是拿与不拿，而拿与不拿正是我们解这一类问题的突破口。

分析：其实也是一个组合解，只不过这个组合有一定限制，把不符合条件的组合解过滤掉以后，再计算总价值，找到最大总价值就可以了。

例题：

|  |  |  |  |
| --- | --- | --- | --- |
| id | 0 | 1 | 2 |
| weight | 1 | 2 | 3 |
| value | 6 | 10 | 12 |

思考：不能使用贪心算法，每次先拿单位重量价值最高的那个物品，举反例就可以推翻贪心算法求解的合理性。

2、我们定义“状态”，寻找“状态转移方程”。

“状态”：在[0,..,i] 这些物品中，限制最大重量是 c 的时候的最大总价值。所以这个状态的函数有两个自变量，一个是 i 另一个是 c。

对于新添加的一个物品来说，我们对它的考虑其实就只有“拿”和“不拿”，从这个角度出发，我们可以很快写出“状态转移方程”：F(i,c) = max[F(i-1,c),F(i-1,c-w(i))+v(i)]，注意，考虑这个物品能够放入背包之前，我们**先要保证这个物品的容量是小于当前能够承受的最大容量的**。

因此，在我们使用“记忆化搜索”实现的时候，使用的缓存，就应该是一个二维数组。

3、下面我们自底向上通过填表来完成这个问题：容量为 5 的时候。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| id\C | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | 6 | 6 | 6 | 6 | 6 |
| 1 | 0 | 6 | 10 | 16 | 16 | 16 |
| 2 | 0 | 6 | 10 | 16 | 18 | 22 |

思考：其实我们填写每一行的时候，只参考了上一行的结果，这一点对于我们思考这个问题和用代码实现来说都是非常重要的。

4、学习完本节的心得：本质上，计算还是帮我们做了遍历搜索，只不过这个搜索最优值的方向是我们程序员输入的算法来实现的，看起来不那么暴力而已，所以是不折不扣的人工智能。

5、能够用小的测试用例来理解 01 背包问题的本质可以帮助我们的编码实现。

9-6 “0-1 背包”问题在空间复杂度上的优化（6题）

原问题的时间复杂度是：O(n\*C)，空间复杂度是：O(n\*C)。我们着眼从空间复杂度上进行优化。

从“状态转移方程”我们可以看出如下优化的方向：

1、优化1（空间复杂度降到O(2\*C)）：从空间复杂度上进行优化，注意到每一行的取值只与上一行相关，即第 1 行我们只处理偶数行，第 2 行我们只处理奇数行；

2、优化2（空间复杂度降到O(C)）：我们又注意到，每一行的取值只与它前面的值有关，因此我们可以从右边到左边来填表，核心代码只有 10 行左右，并且，通过一层判断，循环可以提前结束；

3、0-1 背包问题的变种，搜索一下背包九讲是个什么鬼：

（1）完全背包问题：每个物品可以无限使用；

（2）多重背包：

（3）多维费用：

（4）物品间加入约束：（1）互相排斥（2）互相依赖

9-7 面试中的 0-1 问题（6题）

1、下面的例题 LeetCode 第 416 题其实是非常典型的背包问题。

* 例题1：LeetCode 第 416 题 Partition Equal Subset Sum

题目要求：Partition Equal Subset Sum。Given a non-empty array containing only positive integers, find if the array can be partitioned into two subsets such that the sum of elements in both subsets is equal.

题目难度：**中等**。

中文网址：https://leetcode-cn.com/problems/partition-equal-subset-sum/description/

英文网址：https://leetcode.com/problems/partition-equal-subset-sum/description/

求解关键：这个问题其实是一道非常典型的背包问题。我们把这一堆数分成两个部分，这两个部分的总和相等。因此这个总和一定得是偶数。此时的背包问题一定得把背包填满。我们来思考一下“状态”是什么？“状态转移方程”又是什么？

“状态”：从索引为 [0,...,i] 这些数中，存在一个组合，它们的和为某个数值。

“状态转移方程”：F(i,c) = F(i-1,c) || F(i-1,c-num[i])。看起来，似乎还是一头雾水，我们在实现的时候，要注意递归终止条件的设置，这样就能轻松解决这个问题了。

这个解法的时间复杂度是 O(n\*sum/2)，n 表示有 n 个数，一个一个加进来，想想“0-1”背包问题是不是也是这样解决的？

另外，从题目中给出的约束条件可以知道，我们的这个算法是在计算机可以接受的范围内。

我的解答：还是两种写法：（1）递归+缓存=记忆化搜索（2）动态规划

参考资料：

思考总结：使用“动态规划”解决这个问题的时候，我们会看到 2 个循环，不是双重循环。

2、下面给出 5 个练习：

* 练习1：LeetCode 第 322 题 Coin Change

题目要求：零钱兑换。

题目难度：**中等**。

中文网址：https://leetcode-cn.com/problems/coin-change/description/

英文网址：https://leetcode.com/problems/coin-change/description/

求解关键：

dp[11] = 3

dp[12] = dp[11] + 1

或者 dp[10] + 2

或者 dp[7] + 1

以上三者之中的最小者

我的解答：

参考资料：https://blog.csdn.net/asd136912/article/details/79080693

思考总结：

* 练习2：LeetCode 第 377 题 Combination Sum IV

题目要求：Given an integer array with all positive numbers and no duplicates, find the number of possible combinations that add up to a positive integer target.

题目难度：**中等**。

中文网址：https://leetcode-cn.com/problems/combination-sum-iv/description/

英文网址：https://leetcode.com/problems/combination-sum-iv/description/

求解关键：

我的解答：

参考资料：

思考总结：

* 练习3：LeetCode 第 474 题 Ones and Zeroes

题目要求：一和零。在计算机界中，我们总是追求用有限的资源获取最大的收益。现在，假设你分别支配着 m 个 0 和 n 个 1。另外，还有一个仅包含 0 和 1 字符串的数组。你的任务是使用给定的 m 个 0 和 n 个 1 ，找到能拼出存在于数组中的字符串的最大数量。每个 0 和 1 至多被使用一次。

题目难度：**中等**。

中文网址：https://leetcode-cn.com/problems/ones-and-zeroes/description/

英文网址：https://leetcode.com/problems/ones-and-zeroes/description/

求解关键：

我的解答：

参考资料：

思考总结：

* 练习4：LeetCode 第 139 题 Word Break（这个问题被很多大公司采用，望留意）

题目要求：单词拆分。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。

题目难度：**中等**。

中文网址：https://leetcode-cn.com/problems/word-break/description/

英文网址：https://leetcode.com/problems/word-break/description/

求解关键：

我的解答：

参考资料：

思考总结：

* 练习5：LeetCode 第 494 题 Target Sum

题目要求：目标和。给定一个非负整数数组，a1, a2, ..., an, 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目标数 S 的所有添加符号的方法数。

题目难度：**中等**。

中文网址：https://leetcode-cn.com/problems/target-sum/description/

英文网址：https://leetcode.com/problems/target-sum/description/

求解关键：

我的解答：

参考资料：

思考总结：

9-8 最长上升子序列问题（2题）

最长上升子序列问题（Longest Increasing Subsequence），我们不要求子序列中的元素是连续的。

* 例题1：LeetCode 第 300 题 Longest Increasing Subsequence

题目要求：最长上升子序列。给出一个无序的整形数组，找到最长上升子序列的长度。例如，给出 [10, 9, 2, 5, 3, 7, 101, 18]，

最长的上升子序列是 [2, 3, 7, 101]，因此它的长度是4。因为可能会有超过一种的最长上升子序列的组合，因此你只需要输出对应的长度即可。

你的算法的时间复杂度应该在 O(n2) 之内。

进阶：你能将算法的时间复杂度降低到 O(n log n) 吗?

题目难度：**中等**。

中文网址：https://leetcode-cn.com/problems/longest-increasing-subsequence/description/

英文网址：https://leetcode.com/problems/longest-increasing-subsequence/description/

求解关键：设置“状态”，这里设置“状态”要注意一点，我们考虑这个状态的时候一定会将最后的这个元素选上。即 LIS(i) 表示以第 i 个数字结尾的最长上升子序列的长度，即我们要求 nums[i] 必须被选取。则状态转移的方程为：前面所有的 LIS 中最大的加 1。最后所求，应该扫描一遍这个 LIS[i] 数组，其中最大的就是我们所求的。

我们以下面的数组为例进行说明：

例1：10，9，2，5，3，7，101，18。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 9 | 2 | 5 | 3 | 7 | 101 | 18 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | **2** | **2** | **3** | **4** | **4** |

最关键的就是填这张表，其实非常简单。最后，我们把整个数组扫描一遍，就找到了最大值。

例2：10，15，20，11，9，101。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 | 15 | 20 | 11 | 9 | 101 |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 2 | 3 | 2 | 1 | 4 |

做了两道题，我们就发现规律了，填表的时候，要遍历前面所有的元素，找出比它小的所有的元素的最大值，把这个最大值 + 1，就是这张表里填的元素值。

注意：填表的时候没有的性质：（1）这个数组不是递增的（2）不是向前找最大的 + 1，所有的数字都要扫一下。

我的解答：

参考资料：

思考总结：

**LIS 问题的时间复杂度可以到 O(nlogn)，可以参考算法时空谢勰老师的讲解。**

* **练习1：LeetCode 第 376 题 Wiggle Subsequence**

题目要求：A sequence of numbers is called a wiggle sequence if the differences between successive numbers strictly alternate between positive and negative. The first difference (if one exists) may be either positive or negative. A sequence with fewer than two elements is trivially a wiggle sequence.

题目难度：**中等**。

中文网址：https://leetcode-cn.com/problems/wiggle-subsequence/description/

英文网址：https://leetcode.com/problems/wiggle-subsequence/description/

求解关键：

我的解答：

参考资料：

思考总结：

9-9 LCS，最短路、求动态规划的具体解以及更多（1题）

1、最长公共子序列问题（Longest Common Sequence）

LCS 问题是典型的处理字符串的方法，思路是：从后向前考虑，得到状态转移方程，画图画出递归树结构。实现“记忆化搜索”和“动态规划”的解法。

2、dijkstra 单源最短路径算法其实也是动态规划，仔细体会一下；

3、思考这样一个问题，在我们求解的动态规划问题中，我们如何通过反向查找得到一个具体解，特别地，得到所有具体解：

（1）LIS 问题中：

4 一定是 3 + 1 的解：

3 一定是 2 + 1 的解：

于是一定可以反向得到具体解。

（2）01背包问题也可以反向得到具体解，

4、刘宇波老师建议在 LeetCode 上（搜索动态规划这个标签）做更多关于动态规划的问题，见得多了，能力也就提升了，这个思维的过程是很有意思的。