520_Week 05 学习总结

整个一周的内容都是围绕动态规划展开的。事实上，我在这周之前完全没有接触过任何动态规划相关内容，所以非常感谢超哥带我打开之前一直认为很神秘的世界……

但是写这篇总结的时候，我相信我已经至少入门了，而且掌握了一些题型的解法。

这篇文章有参考”五分钟学算法”公众号中有关动态规划的相关内容，很多地方和超哥讲的内容差不多，我会将两者进行适当的结合。

在正式介绍动态规划之前，超哥回顾了分治，回溯和递归。在课上讲到，“动态规划”和前面三者其实没有本质区别，都是把问题进行拆分，寻找规律。比如分治其实是递归的特殊形式，很多题目子问题都是有自相似性的，关键在于怎么把大问题分解成小的子问题。

养成数学归纳法思想，也就是先把基础条件想明白然后解决，比如 n=1 怎么样，n=2 的时候又怎么样，然后推导到 n 和 n+1 的情况。数学归纳法的思想好比是爆竹，想要整串都能爆炸，我们需要推导所有的最后都能爆炸。

动态规划具体是什么意思？有一个例子其实很生动。比如我们在纸上写了很多个”1+1+1+1+1…”，第一眼看过去，谁都不能知道到底写了多少个 1.但是如果我们已经计算好了比如之前一共写了 8 个 1，那么只要再写一个 1，我们马上就知道现在一共写了 9 个 1，这就是动态规划。**动态规划是一个能够让我们“记忆”之**

前的成果并且把它们运用在后面的运算中的一个方法，能够让程序“记住”之前所得到的答案。动态规划可以说是用空间换时间的一种方式。

对于一个动态规划问题，我们往往需要从两个方面进行考虑，即：**找出问题之间的联系**以及**记录答案**。这里的难点实际是找出问题之间的联系，因为记录答案可以用比较简单的数据结构顺带完成。

解决动态规划问题一般需要四个步骤：

- 问题拆解，找到问题之间的具体联系
- 状态定义
- 递推方程推导
- 代码实现

上述步骤中前两步是最关键的。如果前两步骤顺利完成，后面递推方程的推导是非常简单的。尤其对于面试题，只要找到了合适的状态定义，递推方程往往不会很难，竞赛的话递推方程就可能比较难了。

状态定义其实是需要思考在解决一个问题的时候，我们都做了什么事情，然后得出了什么样的答案。对于这个问题，当前问题的答案就是当前的状态。经过了前面两句话的拆解，可以发现两个相邻的问题的联系其实是：

后一个问题的答案 = 前一个问题的答案+1

这里，状态的每次变化就是+1.

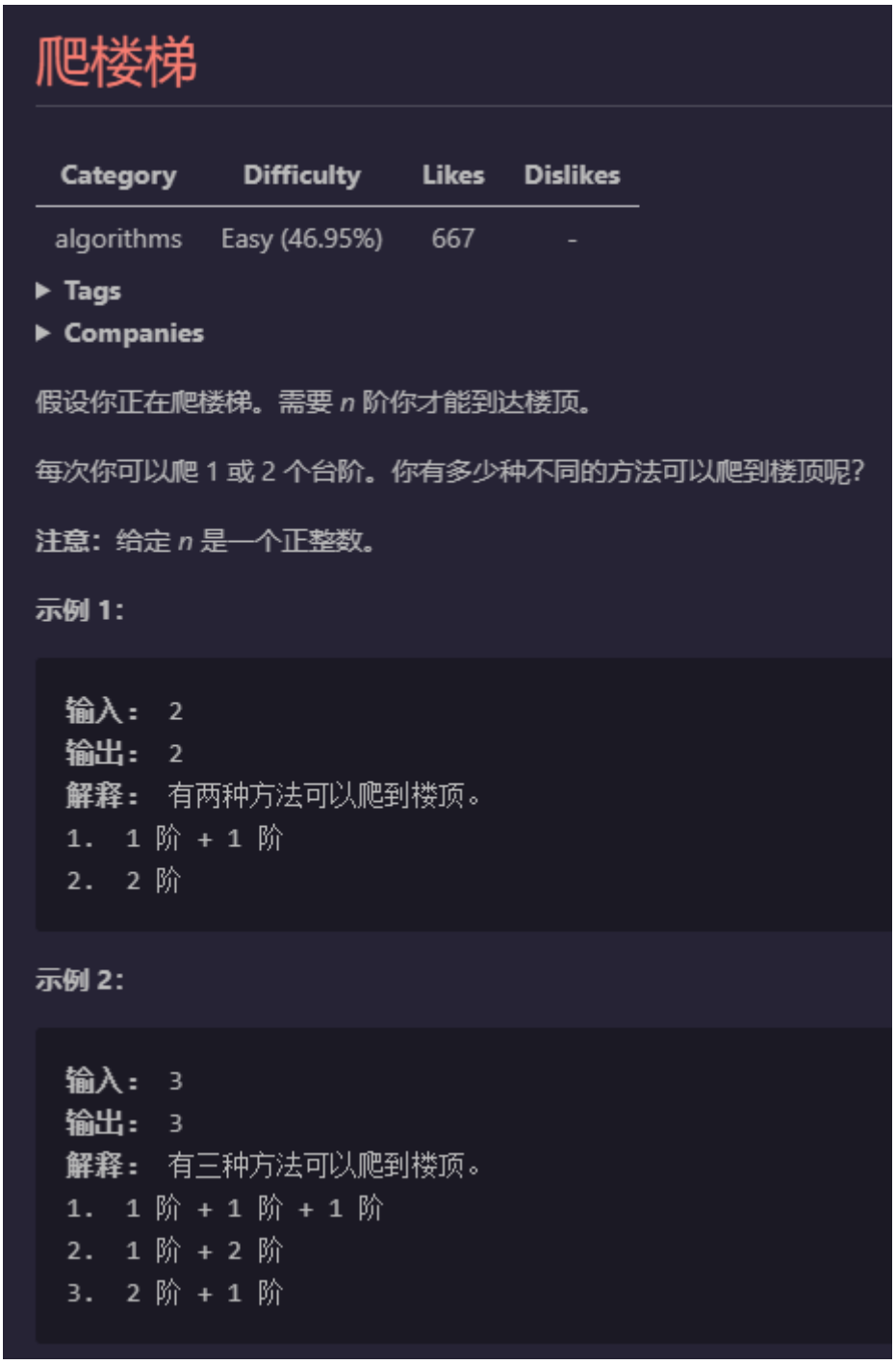
定义好了状态递推方程就可以变得非常简单。

动态规划的四步其实是互相递进的，状态的定义(第二步)离不开问题的拆解(第一

步)、递推方程推导(第三步)离不开状态的定义(第二步), 最后的代码实现(第三步)的核心其实就是递推方程(第四步)。

LeetCode 上面有一些很经典的问题.

第 70 题爬楼梯。



爬楼梯

Category	Difficulty	Likes	Dislikes
algorithms	Easy (46.95%)	667	-

▶ **Tags**

▶ **Companies**

假设你正在爬楼梯。需要 *n* 阶你才能到达楼顶。

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?

注意: 给定 *n* 是一个正整数。

示例 1:

```
输入: 2
输出: 2
解释: 有两种方法可以爬到楼顶。
1. 1 阶 + 1 阶
2. 2 阶
```

示例 2:

```
输入: 3
输出: 3
解释: 有三种方法可以爬到楼顶。
1. 1 阶 + 1 阶 + 1 阶
2. 1 阶 + 2 阶
3. 2 阶 + 1 阶
```

我们可以按照上面的四步进行分析：

1) 问题拆解

我们应当习惯用自底向上的思路方式去做（超哥推荐的），也就是从终点开始想。如果我们要到达第 n 个台阶，按照题意我们会先到达第 n-1 或者底 n-2 级台阶。因此第 n 个问题可以拆解成第 n-1 和 n-2 个子问题，第 n-1 和 n-2 又可以继续往下拆，直到第 0 个问题，也就是上楼梯的起点。

2) 状态定义

在问题拆解的时候已经提到了，第 n 个楼梯会和第 n-1 和第 n-2 个楼梯有关联。而两者具体的联系是什么呢？可以这样思考，第 n-1 个问题中的答案其实是从起点到达第 n-1 个楼梯的路径总数，n-2 同理。而从第 n-1 个楼梯可以到达第 n 个台阶，从第 n-2 个也可以。而且从第 n-1 到达第 n 级台阶与从 n-2 到达第 n 级台阶，这两者之间的路径不可能有重复。因此我们可以把第 i 个状态定义为："**从起点到达第 i 个楼梯的路径总数**"，状态之间的联系其实是相加的关系。

3) 递推方程

只要有了正确靠谱的状态定义，递推方程并不难得到。

因为在"状态定义"中我们已经定义好了状态，也知道第 i 个状态可以由第 i-1 个状态和第 i-2 个状态通过相加得到，因此递推方程可以得到：

$$dp[i]=dp[i-1]+dp[i-2]$$

4) 代码实现

在递推方程中可以看到，我们需要有一个类似"空集"的初始值来方便我们计算。这个起始位置是不用考虑真正用于计算的第一个情况的。所以这里的爬

楼梯问题的初始值是不需要移动的位置：dp[0]=0，第 1 层楼梯只能从起始位置到达，因此 dp[1]=1，第 2 层楼梯可以从起始位置和初始位置到达，因此 dp[2]=2，有了这些初始值，后面就可以用递推方法得到。

注意，**在代码实现的时候，可以不需要和第二步分析状态定义的时候用自底向上的方式**。

```
//方法一，用常规的动态规划去分析
if (n <= 2) return n;

int[] dp = new int[n+1]; //多开一位，考虑起始位置
dp[0]=0; dp[1]=1; dp[2]=2;
for (int i = 3; i <= n; i++) {
    dp[i] = dp[i-1] + dp[i-2];
}
return dp[n];
```

第二道题，LeetCode120，三角形最小路径和

三角形最小路径和

Category	Difficulty	Likes	Dislikes
algorithms	Medium (62.20%)	247	-

▶ Tags
▶ Companies

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。

例如，给定三角形：

```
[
     [2],
    [3,4],
   [6,5,7],
  [4,1,8,3]
]
```

自顶向下的最小路径和为 11（即，2 + 3 + 5 + 1 = 11）。

说明：

如果你可以只使用 O(n) 的额外空间（n 为三角形的总行数）来解决这个问题，那么你的算法会很加分。

用四个步骤进行分析:

1) 问题拆解

这里要求出最小的路径和， 路径是这里分析的重点，路径是由一个个元素组成的，和之前爬楼梯类似，[i][j]可以看做是位置元素。经过这个位置元素，肯定要经过[i-1][j]或者[i][j-1]，因此经过一个元素的路径可以通过这个元素上面的一个或者两个元素的路径的和得到。

2) 状态定义

状态的定义一般会和问题需要求解的答案联系在一起，这里其实有两种方式，一种是考虑路径从上到下，另一种是考虑路径从下到上。因为元素本身不会变，所以路径的方向不同也不会影响最后求得的路径和，如果是从上到下，你会发现，在考虑下面元素的时候，其实元素的路径只会从[i-1][j]获得，每行当中的最后一个元素的路径只会从[i-1][j-1]获得，但是中间的所有元素的值的获得有规律可寻么？没有，所以从上到下很难实现一个统一的公式。

因此我们考虑从下到上的方式，状态的定义就变成了“**最后一行元素到当前元素的最小路径和**”，对于[0][0]这个元素来说，它的内容就是我们的最终答案。

3) 递推方程

已经定义了状态，所以可以得到递推方程:

dp[i][j]=Math.min(dp[i+1][j],dp[i+1][j+1])+triangle[i][j]

4) 代码实现

这里初始化时，我们需要将最后一行的元素填入到状态数组中，然后就是按照前面的分析策略，从下到上计算即可。

```java
//方法一，普通的动态规划的分析
int n = triangle.size();

// dp[i][j]表示最后一行元素到当前元素的最小路径和
int[][] dp = new int[n][n];

//获取最后一行
List<Integer> lastRow = triangle.get(n-1);

// 把最后一行的所有元素放入到二维数组 dp的最后一行
for (int i = 0; i < n; ++i) {
    dp[n-1][i] = lastRow.get(i);
}

//从dp的倒数第二行开始遍历
for (int i = n-2; i >= 0 ;--i) {
    //当前遍历到的行，用于后面的triangle[i][j]的取值
    List<Integer> row = triangle.get(i);
    for (int j = 0; j < row.size(); ++j) {
        dp[i][j] = Math.min(dp[i+1][j], dp[i+1][j+1]) + row.get
    }
}
return dp[0][0];
```

事实上在看了很多道动态规划的题目之后我有这样的发现，虽然动态规划理论上主要是通过空间换时间，但是很多时候，问题的维度是二维的时候，往往可以考虑优化成一维空间来解决。而一维空间的问题，往往可以优化成常数级空间来解决。

曾经在 72 题解中 LeetCode 题解中看到某位同学使用动态规划对自己微信推送进行修改的经历，突然发现这种情景竟然可以如此切合实际……