24 | 动态规划（上）：只需四步，搞定动态规划算法设计

2020-03-12 胡光

人人都能学会的编程入门课 进入课程 >

讲述：胡光

时长 18:03 大小 16.55M

你好，我是胡光，欢迎回来。

上节课呢，我们学习了递推算法的一般求解步骤：先是定义递推状态，然后推导递推公式，最后是程序设计与实现。并且为了顺利完成递推算法，还介绍了在推导递推公式中的重要指导思想“容斥原理”的相关内容。

递推算法解决的主要类型问题之一，就是计数类问题。就像上节课我们提到的，求 n 个月以后的小兔子数量，求拼凑钱币的方法总数，还有更早之前学习的，求前 n 个数字二进表示中 1 的个数，等等，这些都是计数类问题。

而在递推算法中，还有一类不同于计数类问题，它是求解最优化解的问题的算法，这类算法有一个专有名称，叫做：动态规划。这就是我们今天要学习的，递推算法中的一个子集算法，动态规划算法。

初识：数字三角形问题

想了解什么是动态规划算法，咱们得先从一个叫做“数字三角形”的简单的动态规划问题开始。数字三角形这个问题很简单，这里我给出了一个由数字组成的 6 层三角形，如下图所示：

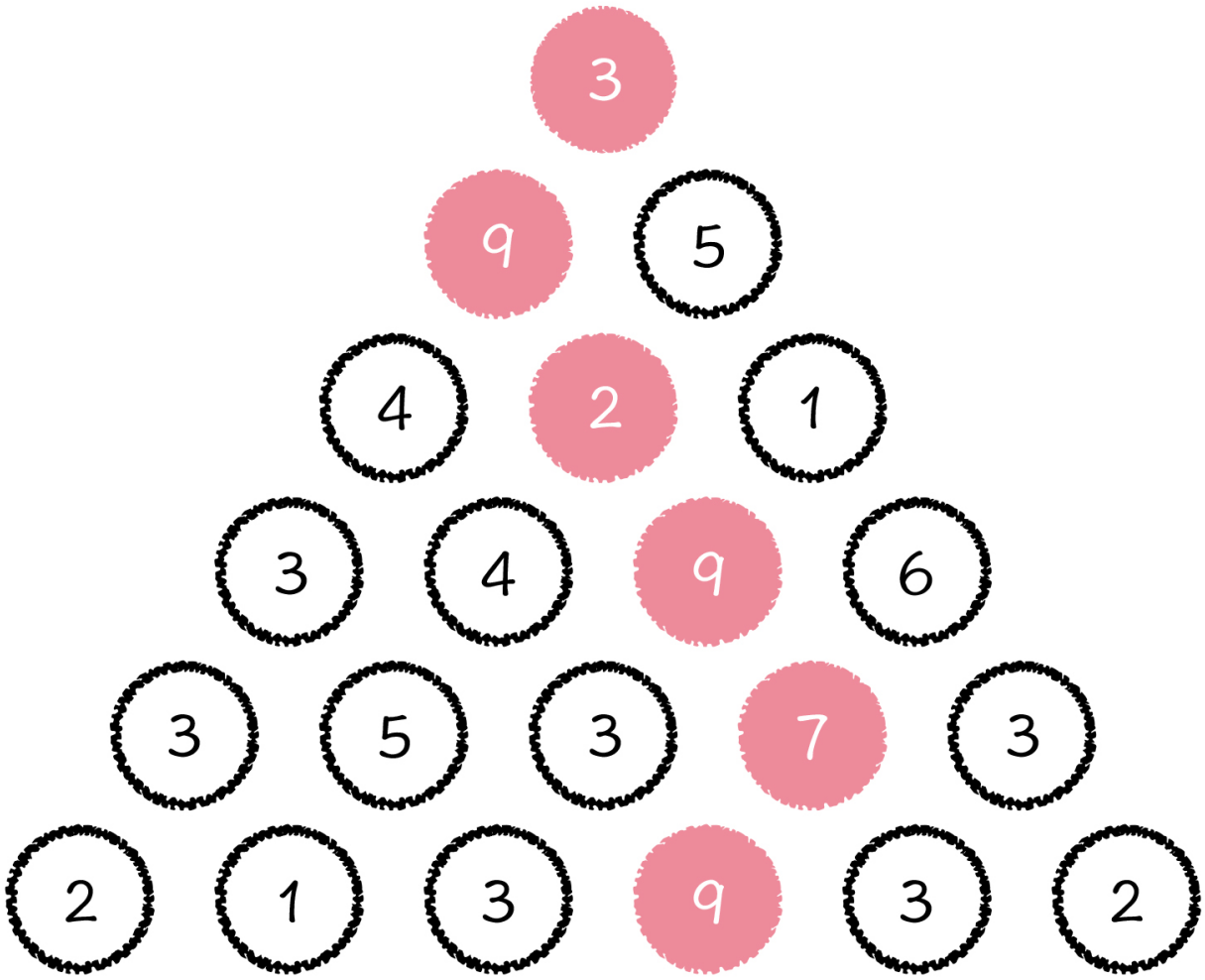

图1： 数字三角形结构示意图

由上到下，第 i 层由 i 个数字组成，目标从第 1 层开始，每次只能向下走到相邻的两个节点，求走到最后一层路径上面数字的最大和值是多少。就像图中标红的一条线路，就是路径和值最大的一条路线，和值为 39。如果给你的是一个 n 层的数字三角形，你该如何解决这个问题呢？

从数学归纳法思想出发，如果我们已知到第三层所有点的最大值，那么我们就可以计算得到起始点到第四层每一个的路径最大和值。如图 2 所示：所有绿色节点和蓝色节点，就是已经求出来的，起始点到其路径最大和值的点。其中的数字是根据 图 1 中的数字三角形计算所得，比如第 2 层的 12，是由图 1 中第 1 层的 3 与原所在位置的 9，相加之和的结果。

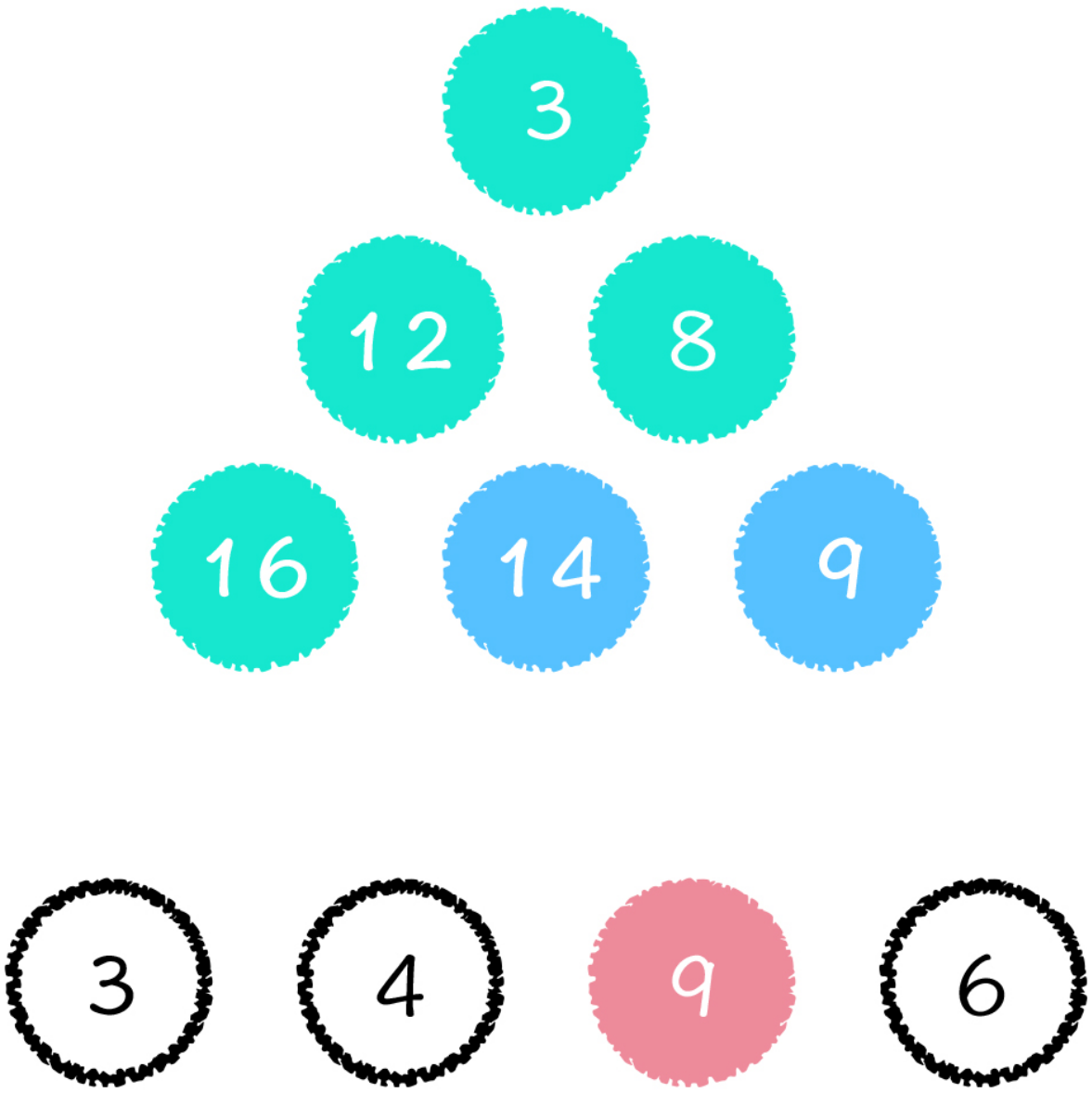

图2： 数学归纳法求解示意图

从图 2 中可知，如果想求从起始点到红色的点，也就是第 4 行数字 9 点的路径最大和值，那么根据数字三角形的规则，我们只能从图中的两个蓝色点转移到红色点。那究竟选择从哪个点走到红色点呢？当然是选择其中和值较大的了，也就是从和值为 14 的点转移到红色点，得到的就是起始点到红色点的路径最大和值。

我们来总结一下上述这个过程，若我们已知从起始点到第 i - 1 层上每个点的路径最大和值，那我们又是怎么得到从起始点到第 i 层上每个点的路径最大和值呢？请看下图：

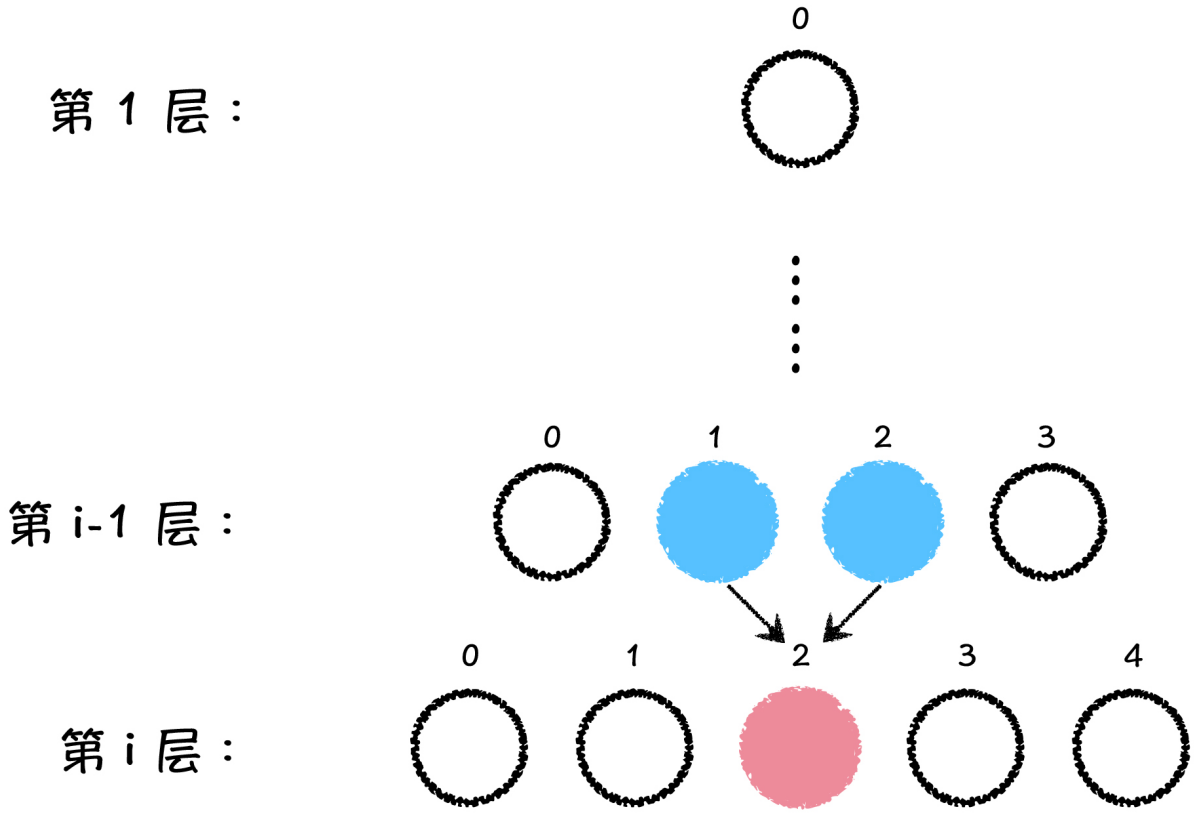

图3： 由第 i-1层推导第 i 层示意图

如图所示，我们给每一层的节点，从左向右，从 0 开始依次编号，那么第 i 行的第 3 个点对应的坐标就是 (i, 2) 点。从第 1 层的点想要到达红色 (i, 2) 点，可以通过 (i - 1, 1) 点到达，或者通过 (i - 1, 2) 点到达。在已知从起始点到第 i-1 层上每个点的路径最大和值的前提下，从第 1 层到 (i, 2) 点的最大和值，就是在 (i - 1, 1) 和 (i - 1, 2) 这两个值中，选择一个路径和值最大的，然后转移到 (i, 2) 点，即为第 1 层到 (i, 2) 点的路径最大和值。

所以，我们基本可以确定一件事情了，如果我们要是知道第 1 层 到 i - 1 层的每个点的路径最大和值，那就很容易求得到第 i 层每个点的路径最大和值，从而推导出 i + 1 层、i + 2 层等等的路径最大和值，直到最后一层。

又因为，我们已知第 1 层到第 1 层每一个点的路径最大和值，就是起始点原本的值，所以沿着上面这个思路，就可以按照层序，来求解第一层到每一层的每个节点的路径最大和值了。

仔细体会一下，上面这个题目的推导过程，有没有点儿我们前面说的数学归纳法思想以及递推算法的意思？你会发现，岂止是有点儿，简直如出一辙。这就是我们所说的，递推算法中那类求解最优化问题的方法，动态规划。

下面我们就正式来介绍一下动态规划问题的求解步骤。

动态规划算法的四步走

关于动态规划，也被简称为：DP(dynamic programming)，它的问题类型非常的庞杂。如果按照问题类型来进行划分，可以分成：线性 DP、区间 DP，树型 DP，数位 DP，概率 DP 等等。说到动态规划中的概念呢，又有什么：最优子结构，重叠子问题，无后效性等等。这些都是让新手听起来特别摸不到头脑的总结性词汇。

但是你也不用着急犯晕，我们知道，任何总结，都来源于观察。所以今天，我想让你掌握的，不是这些前人总结的词汇概念，而是一套观察、学习动态规划的方法。

这套方法分为四个步骤，它会使得你学习动态规划算法的过程事半功倍。如果你按照我的方法，进行了若干种动态规划问题学习以后，再找来一些其他资料，看看今天我跟你说的动态规划中的概念名词，你会对动态规划有一个更具体的理解。

那这个方法到底是哪四个步骤呢？其实就是：**状态定义，状态转移方程，正确性证明，以及程序设计与实现**。同时，它们也分别代表了学会一个动态规划问题的四个方面。

1. 状态定义

首先我们从状态定义讲起，提到状态定义，你应该不会陌生，上节课我们已经说过递推问题的确定递推状态，其实二者是一样的，都是一个有明确语义信息的数学符号。

理解一个动态归划问题的状态定义，是理解其解法的第一步，也是最重要的一步。如果你在往下进行推导的时候，发现进行不下去了，那往往就是状态定义有问题，这时你就需要回到这个第一步，琢磨琢磨新的状态定义了。

并且，我们一直在强调，对于动态规划的状态定义，不仅仅是要一个数学符号，还要一个明确的语义信息，你的理解可能是：不同的语义信息，对应的不就是不同的数学符号么？那今

天，我们就用同一个数学符号，表示不同的语义信息，在接下来的求解过程中，你会发现这两种不同的语义信息，所衍生出来的后续步骤过程，是完全不同的。

回到前面说的数字三角形问题，我们可以作出两种状态定义：

第一种状态定义：dp[i][j] 代表从起始点，到 (i, j) 点的路径最大值。

第二种状态定义：dp[i][j] 代表从底边的某个点出发，到 (i, j) 点的路径最大值。

为了后续讲解方便，我们假设所有坐标都是从 1 开始的，也就是第一行第一个点的坐标是 (1, 1)。你会发现，这两种状态定义，数学符号都是 dp[i][j]，而含义却完全相反，一个是从顶向下走，一个是从底向上走。对于第一种状态定义，如果数字三角形有 n 层的话，问题所求的最大值，就是在最后一层 dp[n] 中的某个值。而第二种状态定义，问题所求的最大值最终会存储在 dp[1][1] 这个状态值中。

2. 状态转移方程

看完了数字三角形问题的两种状态定义以后，下面就来讲讲状态转移方程。动态规划的状态转移方程，其实就是递推问题中所说的递推公式，只是从名字上更符合动态规划问题的情况。

状态转移，就是状态之间的转移，每一个状态的含义，在状态定义中规定的明明白白，而状态与状态之间的转移方式，是需要根据具体的问题以及具体的状态定义，进行具体分析。

根据刚才作的两种状态定义，我们可以分别画出来这样两种状态转移的方向：

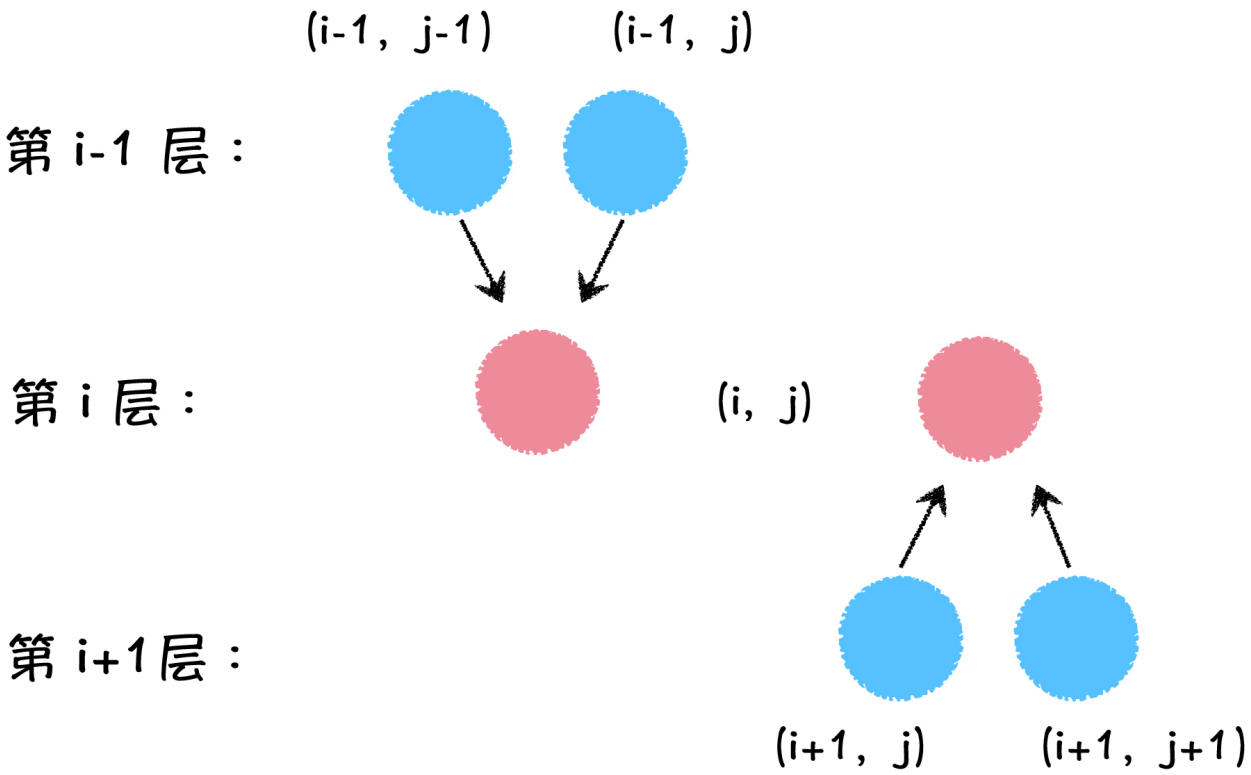

图4： 两种状态转移示意图

如图所示，我以左边是第一种状态定义下的状态转移方向为例，来说明它是如何转移的。首先，它是自上向下转移的，所以想要求得 dp[i][j] 的值，我们需要知道 dp[i - 1][j - 1] 和 dp[i - 1][j] 的值。因为按照“走向下个相邻两点”的规则，只有 (i - 1, j - 1) 和 (i - 1, j) 这两个点，才能能走到 (i, j)，也就是我们讲到的转移到 (i, j) 点。右边的第二种状态定义转移过程和左边的一样，只是移动方向不一样而已。

所以，根据两种状态定义，我们可以分别列出这两种状态转移方程：

第一种状态转移方程：dp[i][j] = max(dp[i - 1][j - 1], dp[i - 1][j]) + val[i][j]

第二种状态转移方程：dp[i][j] = max(dp[i + 1][j], dp[i + 1][j + 1]) + val[i][j]

两种转移方程，都是在能够转移到 (i, j) 点的状态值中选择一个较大值，再加上 (i, j) 原本的数值 val[i][j]，就是各自起始点到达 (i, j) 点的路径最大值，也就是两种状态定义下的 dp[i][j] 的值。

到这里，你可以看出，**状态定义不一样，直接导致我们的状态转移方程就不一样。所以，虽然是相同的数学符号，定义的含义不同，就会造成后续的解法不同，同时也意味着解决问题**

的难度不同。

这也就是很多同学在一开始学习动态规划算法的时候，总喊着不明白状态转移方程，而我会告诉他们的是，你不是不明白状态转移方程，你是不明白状态定义。要解决一个动态规划问题，要从状态定义着手，要学习动态规划算法，也要从状态定义开始学起。

关于状态转移方程这里，我们再来讲一个转移方向的问题。根据数字三角形这个问题的两种状态转移方程，我们可知这代表了两种不同的状态转移方向：第一种是从第一层开始，计算出第二行的所有值，再计算出第三行所有值；而第二种状态转移方向与第一种正好相反。这里我们就要引出动态规划算法中一个最重要的概念“阶段”。

什么是“阶段”呢，可以这样说，状态转移就是从一个阶段转移到下一个阶段。像数字三角形问题中，在第一种转移方式中，起始点的第一层，就是整个转移的第一个阶段，第二层就是整个转移的第二个“阶段”，你会发现转移的时候，只有一个阶段计算完了，才能计算下一个阶段中的状态值。

而在第二种转移方式中，作为起始点的最后一层，才是我们转移的第一个阶段，然后依次由下向上转移，一个阶段接着下一个阶段。

弄清什么是阶段，对于接下来我们证明算法的正确性，有决定性作用。

3. 正确性证明

动态规划算法的第三步，就是证明你推导出的状态转移方程的正确性。关于状态转移方程的正确性证明，借助的就是之前学习中，我们提到过的程序设计中最重要的数学思维：数学归纳法。

根据数学归纳法的三步走，我们试着证明一下第一种状态转移方程是正确的，也就是自上而下的状态转移方式。

第一步，我们已知在这种状态转移方式中，第一个阶段中的所有 dp 值都可以轻松获得，也就是可以很轻松的初始化 dp[1][1] 的值，应该等于 val[1][1] 的值。

第二步，我们假设如果第 i-1 阶段中的所有状态值，我们都正确的得到了。也就是正确的得到了从起始点到 i-1 层中每个点的路径最大和值。那根据状态转移方程：dp[i][j] =

max(dp[i - 1][j], dp[i - 1][j + 1]) + val[i][j] 来说，就可以正确的计算得到第 i 个阶段中的所有状态值。

第三步，两步联立，就可得出结论，所有阶段中的状态值计算均正确。那么，从起始点到底边的路径最大和值，就在最后一个阶段的若干个状态值中。

以上就是我们使用数学归纳法，证明数字三角形问题的第一种状态转移方程正确性的过程。这个过程呢，比较简单，那是因为数字三角形问题本身就不难。当面对更难一些的动态规划问题的时候，将这种证明方法，加入到你学习动态规划算法的过程中，你会收获奇效的。

4. 程序设计与实现

动态规划解题的最后一步，就是程序的设计与实现了。关于数字三角形问题的两种解题方法的代码实现，就作为今天给你留的课后作业题了。

在上一篇递推算法的作业题中，你应该体会到了，对于同样的递推公式，我们不仅可以用循环实现，还可以用递归实现。今天的这两种状态定义方法呢，我只要求你用循环的程序方式实现即可。

当然，我还希望，在你实现出了这两种状态定义方法的程序以后，可以从程序的角度，对两种方法加以评价，并在留言区说说它们的优点和缺点。

课程小结

至此，我们就说完了动态规划算法的完整解题步骤，关于今天的课程呢，希望你记住如下几点：

1. 状态定义，是动态规划算法的重点，无论是解题还是学习，都要从这一步开始。
2. 不同的状态定义，决定了不同的状态转移方程，同时也可能代表了不同的解题难度，所以，学习如何定义优秀的状态很重要。
3. 动态规划中的状态转移顺序，是建立在“阶段”概念之上的，只有本阶段的状态值计算完了，下一个阶段的状态值才能得以计算。
4. 数学归纳法，是证明动态规划状态转移方程正确性的利器，掌握了它，会让你的动态规划学习过程事半功倍！

好了，关于动态规划算法，今天我们就先讲到这里。下一期我们将会使用这两期文章中学习到的技巧，来学习一个稍微有点儿难度的动态规划问题，也算是对我们近期学习效果的一个验证。

再好好看看这两期的内容吧，我是胡光，你要准备好，我们下期见。

上一篇　23 | 深入理解：容斥原理与递推算法

下一篇　25 | 动态规划（下）：背包问题与动态规划算法优化

精选留言 (2)

写留言

徐洲更
2020-03-12

嗯嗯 非常认同老师说的 状态定义是DP最关键的一步，前几天做爬楼梯和零钱兑换题目的时候 就发现同一个状态转移方程，调换了内外循环，就导致结果不一样，后来发现是因为当内外循环变了，状态的含义也就变了。我把这个思考过程记录了下来 http://xuzhougeng.top/archives/difference-between-climb-stairs-and-coin-change-ii#more

展开

作者回复: 完美！赞！

1

胖胖胖
2020-03-15

```
#include <stdio.h>
int main()
{
    int n, i, j;
    int s[50][50];...
```

展开

作者回复: 不错！