# 动态规划详解 第一章

首先让我们看一个例子：
例 1：如下图有一个数字三角阵，请编一个程序计算从顶点至底的某处的一条路径，使该路径所经过的数字的和最大。每一步可沿左斜线向下或右斜线向下走。
7
5 3
4 2 1
2 1 3 7
这个问题的实质实际上是一个有向图中最长路经问题，可以用经典的图论算法或者搜索来解决。
考虑如何用搜索法来解这道题，从第一个点开始扩展，每次扩展 2 个可行节点，直到达到数字三角形的底部，从所有的可行路径中找出最优值，这样做的复杂度是 o(2^n)，当 n 很大的时候，普通的搜索法将不可行。
观察发现，搜索的效率低下很大程度上是因为做了大量的重复运算，**因为对于任何一个节点，从他开始向下拓展的最优值是确定的**，这启发了我们应当充分利用之前的运算结果。
下面我们来进行深入的分析，假如已经走第 I 行第 J 列，此时最大累加和 S[I，J]应选 S[I-1，J]，S[I-1，J+1]中较大者再加上（I，J）处的值 A[I，J]，即下式
S[I，J]=A[I，J]+MAX（S[I-1，J]，S[I-1，J+1]）
所以我们可以从第一行开始，向下逐行推出每一处位置的最大累加和 S，最后从底行的 N 个 S 中选出最大的一个即为所求，这种算法的复杂度为 o(n^2)，比较搜索法，已经大大的降低了，**这种充分利用已经计算出结果的方法，就叫做动态规划。**

通过上面的例子，我们对“动态规划”有了一个初步认识，它所处理的问题是一个“多阶段决策问题”。我们现在对一些概念进行具体定义：
**状态**（State）：它表示事物的性质，是描述“动态规划”中的“单元”的量。如例 1 中的每个节点（指节点处的最大值）都为单个的状态。
**阶段**（Stage）：阶段是一些性质相近，可以同时处理的状态集合。通常，一个问题可以由处理的先后次序划分为几个阶段。如例 1 中的问题，每一行若干节点组成一个阶段。一个阶段既可以包含多个状态，也可以只包含一个状态。描述各阶段状态的变量称为状态变量，例 1 中可用 S[4，j]来表示第四阶段（即第四行）走到第 j 列的最大值，即第四阶段状态变量。
**状态转移方程**：是前一个阶段的状态转移到后一个的状态的演变规律，是关于两个相邻阶段状态的方程，是“动态规划”的中心。
如例 1 中： S[I，J]=A[I，J]+MAX（S[I-1，J]，S[I-1，J+1]）
决策（Decision）：每个阶段做出的某种选择性的行动。它是我们程序所需要完成的选择。如例 1 中 MAX（S[I-1，J]，S[I-1，J+1]）
动态规划所处理的问题是一个“多阶段决策问题”，一般由初始状态开始，通过对中间阶段决策的选择，达到结束状态。这些决策形成了一个决策序列，同时确定了完成整个过程的一条活动路线（通常是求最优的活动路线）

从上面的讲解我们可以发现：**动态规划并不像一种算法，而更像一种思考方式**。

下面，我们来讨论动态规划的应用范围，要确定一个问题是否能用动态规划，需要考虑两个方面：

**一：最优子结构**

即一个问题的最优解只取决于其子问题的最优解，也就是说，非最优解对问题的求解没有影响。我们再来看一个问题：

例二：有 4 个点，分别是 A、B、C、D，相邻两点用两条连线，表示两条通行的道路，给出每条道路的长度。我们定义从 A 到 D 的所有路径中，长度除以 4 所得余数最小的路径为最优路径。求一条最优路径。

很多初学者往往会认为这道题也可以采用动态规划的方法，但实际并不如此，考虑这种情况

假如 A-B 的两条道路分别为 2，3,B-C 的两条道路分别为 1，4。如果采用动态规划，节点 B 的最优值为 2，节点 C 的最优值 2，但际上到达 C 的最优值应该是 0，即 3-1 这条线路。

也就是说，C 的最优取值不是由 B 的最优取值决定的，其不具有最优子结构。

由此可见，并不是所有的"决策问题"都可以用"动态规划"来解决。所以，**只有当一个问题呈现出最优子结构时，"动态规划"才可能是一个合适的侯选方法。**

二：无后效性

即一个问题被划分阶段后，阶段 I 中的状态只能由阶段 I-1 中的状态通过状态转移方程得来，与其他状态没有关系，特别是与未发生的状态没有关系，这就是无后效性。

<<<<<<<<<<<<<<<<<<<<<END>>>>>>>>>>>>>>>>>>

经典题目推荐：

1.最长不下降子序列 O(N^2)版本
2.最长公共子序列
3.最小字母代价树
4.石子合并
5.最优二叉树
6.工作安排
7.背包问题
8.加分二叉树
9.钱币问题

# 动态规划详解 第二章

通过上一章的学习，相信大家对动态规划已经有了一个初步的了解，如果您将上一章的推荐习题全部掌握，那么您可以开始这一章的学习内容了。
这一章，我们将讲解一些动态规划的设计技巧。
相信大家在做动态规划一类题目的时候，往往不容易看出来这道题目是动态规划。其实这并不是您的 IQ 低，几乎所有的初学者都会存在这样的问题，我同样也不例外，不过凭借我超人的天赋，终于总结出来如何看出来某道题是不是动态规划的终极方法：
**做题做题在做题！**在一年半的学习中，我深刻的体会到了：**动态规划是做出来的！**
这个硬道理，没有题目数量的保证，学好动态规划是很困难的。
但是动态规划也并非是无章可循，下面就为大家介绍几种常见的类型：
1．极值问题
这类问题的显著特征就是让您求出最 X 值，并且往往具有最后子结构的性质，因为其模型变化丰富，并且和实际联系紧密，所以在动态规划类问题中占有很大的重量
2．总数问题
3．一类博弈问题
4．某些杂题
一般情况下，动态规划类的问题往往就这几种类型，因此当大家看到以上几种类型的时候，不妨向动态规划的方向作些思考。

学习动态规划的另一个难点就是状态的设计，可以说，只要有了状态，那么决策和阶段也就迎刃而解了。
**一个好的状态，首先要把问题描述清楚，其次转移的时候应当尽量"简单"，这里的简单，主要指状态之间的"距离"**，譬如 A[I]=MAX(A[J]) J<=I 就没有 A[I]=A[I-1]好。
下面让我们看一道题目：
例：排队买票
问题描述：一场演唱会即将举行。现有 N（O〈N〈=200〉个歌迷排队买票，一个人买一张，而售票处规定，一个人每次最多只能买两张票。假设第 I 位歌迷买一张票需要时间 Ti（1〈=I〈=n〉，队伍中相邻的两位歌迷（第 j 个人和第 j+1 个人）可以由前一个人买两张票，也可由后一位买两张票，则另一位就可以不用排队了，则这两位歌迷买两张票的时间变为 Rj，现给出 N，Tj 和 Rj，求使每个人都买到票的最短时间和方法。
因为可以从前一个人买票，又可以从后一个人买票，似乎不符合无后效性的原则，没有办法用动态规划求解。
所以我们不妨采用**加一维**的策略：
设 F（I，**）为到第 i 个人为止所需的最短时间.
设 Ti 为第 I 个人单独买票的时间。
设 Ri 为第 I 个人买 2 张票的时间。
则：状态转移方程分 5 种情况：
F（I，仅买 1 张）=MIN{F（I-1，仅买 1 张），F（I-1，买 2 张代前一位），F

（I-1，不买由前代）}+Ti
F（I，买 2 张代前一位）=MIN{F（I-2，仅买 1 张），F（I-2，买 2 张代前一位），F（I-2，不买由前代）}+Ri
F（I，买 2 张代后一位）=MIN{F（I-1，仅买 1 张），F（I-1，买 2 张代前一位），F（I-1，不买由前代）}+Ri
F（I，不买由前代）= F（I-1，买 2 张代后一位）
F（I，不买由后代）=MIN{ F（I-1，仅买 1 张），F（I-1，买 2 张代前一位），F（I-1，不买由前代）}
注：本题的状态还可以继续优化到 3 种，请大家自己思考。

从上面的例子我们可以看到，当遇到题目的状态无论如何也无法免除后效形时，可以采用**加一维**的方法来解决，有的时候甚至需要加两维甚至三维。

处理动态规划时还有几种技巧：
1. 对于一些需要计算出权函数的题目，不妨把函数的计算放置到动态规划之外，同时还可以尝试对一些权函数进行合并，这类处理权函数的技巧还有很多，相信大家在做题的过程中都会遇到，就不再详细展开了
2．对于某些状态，不妨对约束条件加以**宽松**，往往会起到出其不意的效果。
3．可以采用**循环数组**技术，比较经典的应用就是上一章的数字三角形，因为只和上一层的状态有关，和其他的状态无关，所以不妨只保存上一层的结果，每次计算出新的结果后就把上一层的结果舍弃，这样会让空间复杂度大大降低，是非常有用的一种优化方法。
4．采用**状态压缩**技术，这个优化的应用较为复杂，变形很多，有兴趣的不妨在网上查找相关资料

<<<<<<<<<<<<<<<<<<<<<TO BE CONTINUE >>>>>>>>>>>>>>>>>>>>>>>>

推荐题目：
1．数字三角形（采用循环数组）
2．金明的预算方案（NOIP2006 年提高组试题）
3．基因重组
4．数的划分
5．免费馅饼
6．卡车更新
7．三维最大数字和
8．花店柜台布置
9．积木游戏
10．musical themes（USACO 5.1.3）
11．buy low,buy lower（USACO 4.3.1）
12．Rectangular Barn (USACO 6.1.2)
本章内容看似简单实则很难，仅仅是以上几个题目是不够的，而且几乎所有的动态规划试题都可以作为本章的练习，所以大家在做完上面的题目后不妨多找一些题目练习

# 动态规划详解 第三章

这一章，我们来学习树形动态规划。
动态规划一般来说分为四大类：线性动态规划，区间动态规划，树形动态规划和特殊种类动态规划。
因为线性模型和区间类模型紧密相关，所以一般我们将这两种类型放在一起学习。
树形动态规划和以上两种不同，它是在一个树结构中进行的，因此具有一般性，
而特殊种类动态规划则包含比较广，譬如状态压缩，
博弈以及其他各种问题都可以归纳到这个部分，因为问题比较分散，
所以这个部分就请大家自己学习，可以参考历年国家集训队论文，
对这个部分的讨论十分透彻和深入。

下面就让我们来学习树形动态规划。
首先看一道例题：
例：给定一棵树 T，树中每个顶点 u 都有一个权 w(u)，权可以是负数。
现在要找到树 T 的一个个连通子图使该子图的权之和最大。
分析一下这道题，我们发现，对于节点 A，如果考虑以他为根的一棵子树，
那么它的最优值和他的父节点无关，所以转移方程为：
F[I]=sigma{F[son],当 F[son]>0 时}+w(u)
最终的答案为 F[X]中的最大值。
因为是在一棵树当中进行动态规划，所以不妨用递归的形式来编写。

通过上面的例子，相信大家对树形动态规划有了一个初步的认识，
其实树形动态规划属于动态规划中一个较难的问题，因为约束条件往往比较多，
所以状态的转移并不容易，下面看一个较难的例子：

在一棵树中，每条边都有一个长度值，现要求在树中选择 3 个点 X、Y、Z，
满足 X 到 Y 的距离不大于 X 到 Z 的距离，且 X 到 Y 的距离与 Y 到 Z 的距离之和最大，求这个最大值。
其中顶点个数 N<=2000000
因为数据规模很大，所以考虑 O(N)或者 O(NlogN)的算法。
分析问题，我们发现因为在一棵树中，所以 x,y,z 的路径中必定有一个交点 k，
距离 S=xk+2*ky+kz，我们发现，若要 S 达到最大值，
那么只需要 ky,xk,kz 为以 k 为根的树中三个不在同一子树中的最大的三个距离就可以了，
但是这样的时间复杂度依然为 O(n^2)，继续分析发现，
实际上第一次求出以某个节点为根的距离后，
只需要以这个点为根再 dfs 遍历一遍就可以求出其他的节点距离，所以总的时间复杂度为 O(N)。

再次回味一下这道题的解题思路：**我们首先从数据规模确定了算法的复杂度，**

**并通过对模型的初步分析得到了一些性质，然后通过这些性质得到了新的解题思路，**
**这其实是解树形动态规划乃至其他问题的一个一般性的思路**。
通过对树模型的特殊性分析，可以大大的降低时间复杂度。

最后让我们来看一个很具有发散性的题目：

奶牛成群、土地众多的 FJ 有一个地形狭长的农场，农场被分成了 n 块土地，n 不超过 1000。
这些土地位于一条直线上，并从左到右编号为 1 至 n。每块土地的面积都相同，但是高度不一定相同。每块土地都拥有一个海拔高度值，这个值不超过 1000000。
如果一段相同高度土地的两边都比它低或者是农场的边界，那么这段土地将被称之为"山顶"。
FJ 希望通过搬走泥土来降低某些土地的海拔高度，使"山顶"的数目不超过 k，其中 $1 \le k \le 25$。
在这一前提下，FJ 希望搬运的泥土体积最小，也就是所有的土地减少的高度和最小。

初看上去这道题似乎和树形动态规划没有什么关系，根本没有树结构的样子，
分析一下我们发现，所谓的山峰，就是每个层中不连接的部分，所以从层的角度思考，
显然，每个层之与他的上层和他的下层的有关，与层的长度无关，所以不访化层为点，
以地面为基础节点，每一层不连接"块"为一个节点，用边连接不同层的相邻的点，我们惊讶的发现，这些点构成了一棵树！
进一步分析，发现每次"削平"一个山顶，在树中就是删除一个最叶子节点，
而消除的高度则可以看作是节点的权值，这样，问题就转化到了在树中删除一些节点，
使最终的节点数目不超过 k+1，并且剩余权值最大的问题，复杂度为 o(nk^2)。

多么巧妙的转化！一道看似复杂的问题就这样解决了，**以无限为有限,以无法为有法**，这或许就是算法的魅力。

<<<<<<<<<<<<<<<<<<<<<<<<<<<To be continue>>>>>>>>>>>>>>>>>>
推荐题目：
1. 树的最大连通分支问题
2. 爱心蜗牛
3. 皇后万岁
4. 有线电视网
5. 贪吃九头鸟
6. 寻宝
7. 网络收费(noi2006)