

动规四要素与坐标型DP

4 DP Key Points & Coordinate DP

课程版本 v6.1

主讲 令狐冲



扫描二维码关注微信/微博
获取最新面试题及权威解答

微信: [ninechapter](#)

微博: <http://www.weibo.com/ninechapter>

知乎: <http://zhuoanlan.zhihu.com/jiuzhang>

官网: <http://www.jiuzhang.com>

本节课的学习目的

完整的掌握动态规划问题的分析方法和原理

“入门”动态规划而非“精通”

解决动态规划问题的步骤

1. 判断是否采用动态规划
2. 归类属于哪一型动态规划
3. 按照动规四要素解题

判断是否采用动态规划

三要

1. 求最值
2. 求方案总数
3. 求可行性

三不要

1. 求所有具体的方案（最坏情况下用了**DP**也没有优化效果）
2. 输入数据无序（除了背包类）
3. 暴力算法时间复杂度已经多项式级别（ $2^n \rightarrow n^2$ ）

坐标型

- 二维坐标
- 一维坐标（接龙）

序列型

- 单序列型
- 双序列型
- 划分型

背包型

区间型

博弈型

树型

状态压缩型

独孤九剑 —— 破气式

动态规划四要素：

状态，方程，初始化，答案

递归四要素 vs 动规四要素

动规的状态 **State** —— 递归的**定义**

- 用 $f[i]$ 或者 $f[i][j]$ 代表在某些特定条件下某个规模更小的问题的答案
- 规模更小用参数 i, j 之类的来划定

动规的方程 **Function** —— 递归的**拆解**

- 大问题如何拆解为小问题
- $f[i][j]$ = 通过规模更小的一些状态求 $\max / \min / \text{sum} / \text{or}$ 来进行推导

动规的初始化 **Initialize** —— 递归的**出口**

- 设定无法再拆解的极限小的状态下的值
- 如 $f[i][0]$ 或者 $f[0][i]$

动规的答案 **Answer** —— 递归的**调用**

- 最后要求的答案是什么
- 如 $f[n][m]$ 或者 $\max(f[n][0], f[n][1] \dots f[n][m])$

递归四要素完全对应动规四要素

这也就是为什么动态规划可以使用
“递归”版本的记忆化搜索来解决的原因！

记忆化搜索 vs 多重循环

优点：从搜索转换，~~缺点~~不易写循环条件和顺序非常容易写错

缺点：递归导致 **StackOverflow**，~~优点~~没有递优化余地（滚动数组）

坐标型动态规划

状态：坐标

方程：根据移动规则

初始化：第0行第0列

答案：目标坐标

Triangle

<https://www.lintcode.com/problem/triangle/>

<https://www.jiuzhang.com/solution/triangle/>

让我们用多重循环的重做一下这个题

状态：坐标

方程：从哪儿来

初始化：起点

答案：终点

```
def minimumTotal(self, triangle):
    n = len(triangle)

    # state: dp[i][j] 代表从 0, 0 走到 i, j 的最短路径值
    dp = [[0] * (i + 1) for i in range(n)]

    # initialize: 三角形的左边和右边要初始化
    # 因为他们分别没有左上角和右上角的点
    dp[0][0] = triangle[0][0]
    for i in range(1, n):
        dp[i][0] = dp[i - 1][0] + triangle[i][0]
        dp[i][i] = dp[i - 1][i - 1] + triangle[i][i]

    # function: dp[i][j] = min(dp[i - 1][j - 1], dp[i - 1][j]) + triangle[i][j]
    # i, j 这个位置是从位置 i - 1, j 或者 i - 1, j - 1 走过来的
    for i in range(2, n):
        for j in range(1, i):
            dp[i][j] = min(dp[i - 1][j], dp[i - 1][j - 1]) + triangle[i][j]

    # answer: 最后一层的任意位置都可以是路径的终点
    return min(dp[n - 1])
```

状态：坐标

方程：到哪儿去

初始化：终点

答案：起点

```
def minimumTotal(self, triangle):  
    n = len(triangle)  
  
    # state: dp[i][j] 代表从 i,j 走到最底层的最短路径值  
    dp = [[0] * (i + 1) for i in range(n)]  
  
    # initialize: 初始化终点 (最后一层)  
    for i in range(n):  
        dp[n - 1][i] = triangle[n - 1][i]  
  
    # function: 从下往上倒过来推导, 计算每个坐标到哪儿去  
    # dp[i][j] = min(dp[i + 1][j], dp[i + 1][j + 1]) + triangle[i][j]  
    for i in range(n - 2, -1, -1):  
        for j in range(i + 1):  
            dp[i][j] = min(dp[i + 1][j], dp[i + 1][j + 1]) + triangle[i][j]  
  
    # answer: 起点就是答案  
    return dp[0][0]
```

自顶向下 vs 自底向上

两种方法都可以，你爱用
一个关心从哪儿来，一个关心
我要到哪里去
我比较喜欢**自顶向下**（正着循



DP空间优化的技巧——滚动数组

坐标 DP 的杀手锏

休息 5 分钟

Take a break

后续课程推荐



系统设计班

⌚ 5 日, 11 小时 后开始直播

互动课

查看



全栈开发基础项目课

⌚ 等待下次开课

直播课

查看详情



人工智能集训营

⌚ 6 日, 19 小时 后截止报名

集训营

查看详情



硅谷求职算法1对1集训营

⌚ 11 月 后截止报名

集训营

查看详情

Knight Shortest Path I & II

<https://www.lintcode.com/problem/knight-shortest-path>

<https://www.jiuzhang.com/solution/knight-shortest-path>

<https://www.lintcode.com/problem/knight-shortest-path-ii>

<https://www.jiuzhang.com/solution/knight-shortest-path-ii>

八个方向 vs 四个方向

哪个可以用动态规划，为什么？

动态规划需要有方向性

无方向则有可能产生循环依赖

接龙型动态规划

属于“坐标型”动态规划的一种
题型一般是告诉你一个接龙规则，让你找最长的龙

Longest Increasing Subsequence

<http://www.lintcode.com/problem/longest-increasing-subsequence/>

<http://www.jiuzhang.com/solutions/longest-increasing-subsequence/>

接龙规则：从左到右一个比一个大

Longest Increasing Subsequence

- 将 n 个数看做 n 个木桩，目的是从某个木桩出发，从前向后，从低往高，看做多能踩多少个木桩。
- state: $f[i]$ 表示（从任意某个木桩）跳到第 i 个木桩，最多踩过多少根木桩
- function: $f[i] = \max\{f[j] + 1\}$, j 必须满足 $j < i \ \&\& \ \text{nums}[j] < \text{nums}[i]$
- initialize: $f[0..n-1] = 1$
- answer: $\max\{f[0..n-1]\}$

Longest Continuous Increasing Subsequence II

<https://www.lintcode.com/problem/longest-continuous-increasing-subsequence-ii/>

<https://www.jiuzhang.com/solution/longest-continuous-increasing-subsequence-ii/>

四个方向，依然可以动态规划

Russian Doll Envelopes

<http://www.lintcode.com/problem/russian-doll-envelopes/>

<http://www.jiuzhang.com/solution/russian-doll-envelopes/>

接龙规则：大信封套小信封

Largest Divisible Subset

<http://www.lintcode.com/en/problem/largest-divisible-subset/>

<http://www.jiuzhang.com/solutions/largest-divisible-subset/>

接龙规则：后面的数可以整除前面的数

给一个单词集合，两个单词可以接在一起当且仅当前一个单词的后缀和后一个单词的前缀，能够重叠（至少1个字符），如

World + dog = Worldog

Hello + low = Hellow

may + maybe = maybe

每个单词可以且仅可以使用一次，请问最长接出来的龙的长度多少？

这个题是否可以使用动规规划？

想要精通动态规划？



ALGORITHM

九章算法强化班

🕒 2 周, 3 日后开始直播

直播课

查看详情



动态规划专题班

🕒 1 月 后开始直播

直播课

查看详情

师父领进门，修行靠自身

祝大家都能找到自己理想的工作!
拿到 Offer 记得微信上告诉我哦!