动态规划入门与动规四要素

主讲人 令狐冲

动态规划 Dynamic Programming

简称动规或者DP

是一种算法思想，而不是一种具体的算法

核心思想：由大化小

动态规划的算法思想：大规模问题的依赖于小规模问题的计算结果

类似思想算法的还有：递归，分治法

动态规划 DP vs 贪心法 Greedy

动态规划为了长远的利益会损失当前利益

贪心法永远追求当前利益最大化

动态规划的两种实现方法

1. 记忆化搜索 (使用递归实现)
2. 多重循环 (使用for循环实现)

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

自底向上的动态规划

状态：坐标

方程：到哪儿去

初始化：终点

答案：起点

```python
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]
```

自顶向下的动态规划

状态：坐标

方程：从哪儿来

初始化：起点

答案：终点

```python
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])
```

自顶向下 vs 自底向上

两种方法都可以，爱用哪个用哪个

一个关心从哪儿来，一个关心到哪儿去

动规四要素

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 / sum / 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] … 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！

不同的路径 Unique Paths

https://www.lintcode.com/problem/unique-paths/

https://www.jiuzhang.com/solutions/unique-paths/

求方案总数类 DP 题