

动态规划 (Dynamic Programming)

@M了个J

<https://github.com/CoderMJLee>

<https://space.bilibili.com/325538782>



实力IT教育 www.520it.com





动态规划 (Dynamic Programming)

- 动态规划，简称DP

- 是求解最优化问题的一种常用策略

面试题47. 礼物的最大价值

在一个 $m \times n$ 的棋盘的每一格都放有一个礼物，每个礼物都有一定的价值（价值大于 0）。你可以从棋盘的左上角开始拿格子里的礼物，并每次向右或者向下移动一格、直到到达棋盘的右下角。给定一个棋盘及其上面的礼物的价值，请计算你最多能拿到多少价值的礼物？

输入：

```
[  
  [1,3,1],  
  [1,5,1],  
  [4,2,1]  
]
```

输出：12

解释：路径 $1 \rightarrow 3 \rightarrow 5 \rightarrow 2 \rightarrow 1$ 可以拿到最多价值的礼物

提示：

- `0 < grid.length <= 200`
- `0 < grid[0].length <= 200`

■ 类似的题目

□ [64. 最小路径和](#)

□ [62. 不同路径](#)

■ 假设 $dp[row][col]$ 是走到 $[row][col]$ 位置时的最大价值

■ 你是如何走到 $[row][col]$ 位置的？有2种可能

□ 从 $[row][col - 1]$ 位置往右走

□ 从 $[row - 1][col]$ 位置往下走

		$[row - 1][col]$
	$[row][col - 1]$	$[row][col]$

```
{ 1, 3, 1, 2 },
{ 1, 5, 1, 3 },
{ 4, 2, 1, 4 },
{ 3, 2, 6, 5 }
```

dp	0	1	2	3
0	1	4	5	7
1	2	9	10	13
2	6	11	12	17
3	9	13	19	24

■ 所以 $dp[row][col] = \max(dp[row][col - 1], dp[row - 1][col]) + grid[row][col]$

121. 买卖股票的最佳时机

假设把某股票的价格按照时间先后顺序存储在数组中，请问买卖该股票一次可能获得的最大利润是多少？

输入：[7,1,5,3,6,4]

输出：5

解释：在第 2 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，最大利润 = $6 - 1 = 5$ 。

注意利润不能是 $7 - 1 = 6$ ，因为卖出价格需要大于买入价格。

输入：[7,6,4,3,1]

输出：0

解释：在这种情况下，没有交易完成，所以最大利润为 0。

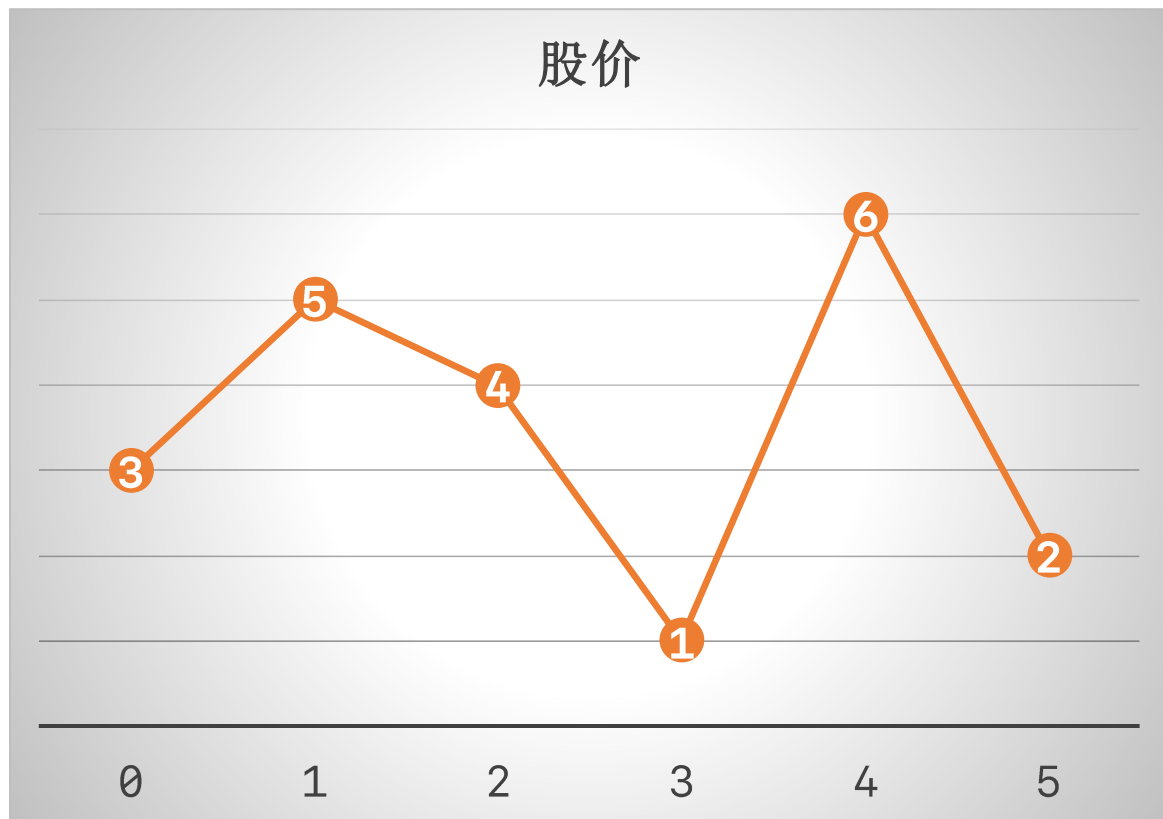
■ 相同的题目

□ [面试题63. 股票的最大利润](#)

■ 时间复杂度： $O(n)$

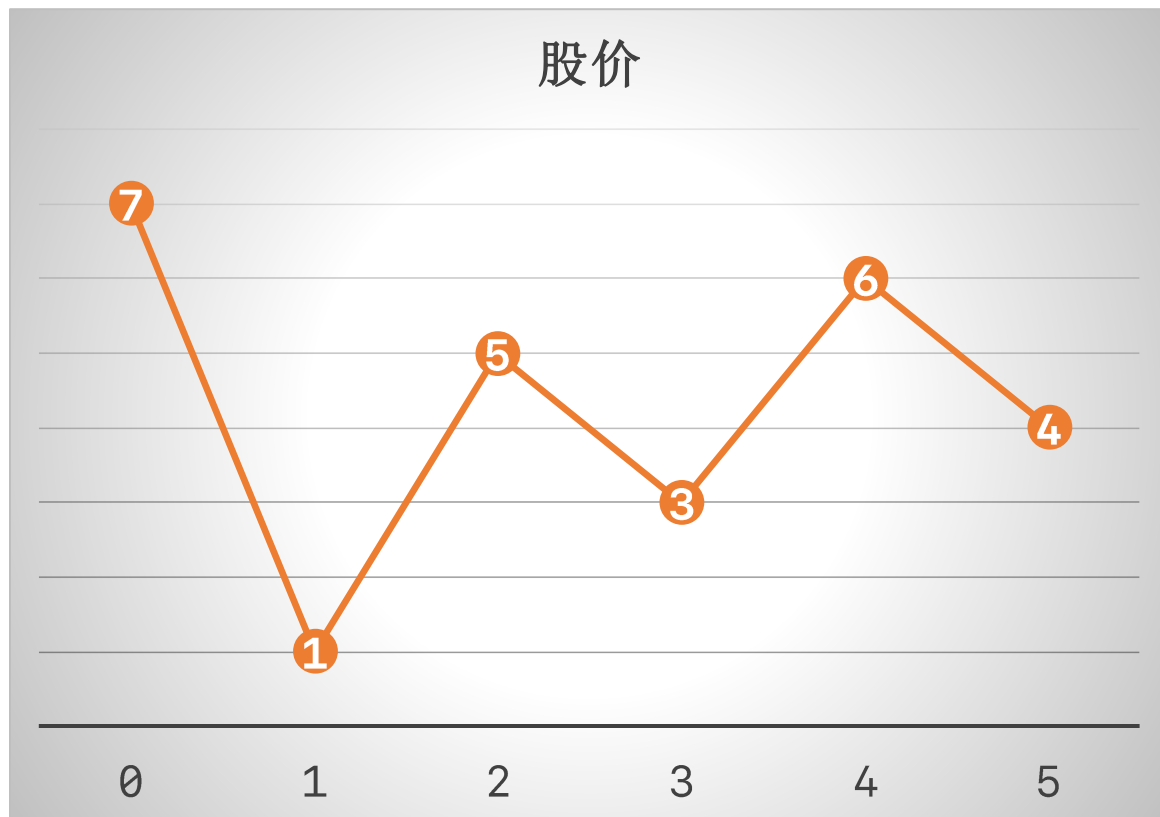
■ 空间复杂度： $O(1)$

0	1	2	3	4	5
3	5	4	1	6	2



卖出价	3	5	4	1	6	2
买入价		3	3		1	1
利润		2	1		5	1

0	1	2	3	4	5
7	1	5	3	6	4



■ 第*i*天买，第*j*天卖的利润是

□ 第*i*~*j*天内，所有相邻两天股价差的和

■ 第1天买，第4天卖的利润是

□ $(6-3)+(3-5)+(5-1) == 6-1 == 5$

相邻两天的股价差				
0~1	1~2	2~3	3~4	4~5
-6	4	-2	3	-2

■ 于是，转化为了求【最大子数组和】的问题

□ 也就是求【最大连续子序列和】的问题

72. 编辑距离

给定两个单词 *word1* 和 *word2*，计算出将 *word1* 转换成 *word2* 所使用的最少操作数。

你可以对一个单词进行如下三种操作：

1. 插入一个字符
2. 删除一个字符
3. 替换一个字符

输入：word1 = "horse", word2 = "ros"

输出：3

解释：

horse -> rorse (将 'h' 替换为 'r')

rorse -> rose (删除 'r')

rose -> ros (删除 'e')

编辑距离算法被数据科学家广泛应用，是用作机器翻译和语音识别评价标准的基本算法。

			j						
				s2[0]	s2[1]	s2[2]	s2[3]	s2[4]	
			dp	0	1	2	3	4	5
i			0						
s1[0]	m	1							
s1[1]	i	2							
s1[2]	c	3							
s1[3]	e	4							

- 假设字符串1 ("mice") 为s1, 它的长度为n1; 字符串2 ("arise") 为s2, 它的长度为n2
- dp是大小为(n1 + 1) * (n2 + 1)的二维数组
- dp[i][j]是s1[0, i)转换成s2[0, j)的最少操作数
- s1[0, i)是由s1的前i个字符组成的子串
- s2[0, j)是由s2的前j个字符组成的子串
- 很显然, dp[n1][n2]就是我们要的答案, 就是s1[0, n1)转换成s2[0, n2)的最少操作数
- 也就是s1转换成s2的最少操作数

				s2[0]	s2[1]	s2[2]	s2[3]	s2[4]	
				j	a	r	i	s	e
			dp	0	1	2	3	4	5
		i	0	0	1	2	3	4	5
s1[0]	m		1	1					
s1[1]	i		2	2					
s1[2]	c		3	3					
s1[3]	e		4	4					

■ 最左上角的 $dp[0][0]$: 代表 $s1$ 的空子串转换为 $s2$ 的空子串的最少操作数

□ 其实就是什么也不用做, 所以: $dp[0][0] = 0$

■ 第0列的 $dp[i][0]$: 代表 $s1[0, i)$ 转换为 $s2$ 的空子串的最少操作数

□ 其实就是删除 $s1[0, i)$ 的所有字符, 所以: $dp[i][0] = i$

■ 第0行的 $dp[0][j]$: 代表 $s1$ 的空子串转换为 $s2[0, j)$ 的最少操作数

□ 其实就是插入 $s2[0, j)$ 的所有字符, 所以: $dp[0][j] = j$

				0	1	2	3	4
			j	a	r	i	s	e
		dp	0	1	2	3	4	5
i		0	0	1	2	3	4	5
0	m	1	1	1	2	3	4	5
1	i	2	2	2	2	2	3	4
2	c	3	3	3	3	3	3	4
3	e	4	4	4	4	4	4	3

■ 如何求出其他位置的 $dp[i][j]$?

□ $dp[i][j]$ 是 $s1[0, i)$ 转换成 $s2[0, j)$ 的最少操作数

□ 可以分4种情况讨论

① 先删除 $s1[0, i)$ 的最后一个字符得到 $s1[0, i - 1)$

□ 然后由 $s1[0, i - 1)$ 转换为 $s2[0, j)$

□ 这种情况下, $dp[i][j] = 1 + dp[i - 1][j]$

② 先由 $s1[0, i)$ 转换为 $s2[0, j - 1)$, 然后在最后插入字符 $s2[j - 1]$, 得到 $s2[0, j)$

□ 这种情况下, $dp[i][j] = dp[i][j - 1] + 1$

③ 如果 $s1[i - 1] \neq s2[j - 1]$, 先由 $s1[0, i - 1)$ 转换为 $s2[0, j - 1)$

□ 然后将 $s1[i - 1]$ 替换为 $s2[j - 1]$, 这种情况下, $dp[i][j] = dp[i - 1][j - 1] + 1$

④ 如果 $s1[i - 1] == s2[j - 1]$, 由 $s1[0, i - 1)$ 转换为 $s2[0, j - 1)$ 后就不用再做任何操作

□ 这种情况下, $dp[i][j] = dp[i - 1][j - 1]$

5. 最长回文子串

给定一个字符串 `s`，找到 `s` 中最长的回文子串。你可以假设 `s` 的最大长度为 1000。

输入: "babad"

输出: "bab"

注意: "aba" 也是一个有效答案。

输入: "cbbd"

输出: "bb"

i	j				
↓	↓				
0	1	2	3	4	5
a	b	b	a	b	a

- 列举出所有的子串，时间复杂度： $O(n^2)$
- 检查每一个子串是否为回文串，每一个子串所需时间复杂度： $O(n)$
- 总共时间复杂度： $O(n^3)$ ，空间复杂度： $O(1)$

		j	b	a	b	a	d
i	dp	0	1	2	3	4	
b	0	T	F	T	F	F	
a	1		T	F	T	F	
b	2			T	F	F	
a	3				T	F	
d	4					T	

■ 动态规划解法

□ 其实是基于暴力法的优化，优化的部分：判断每个串是否为回文串

□ 时间复杂度： $O(n^2)$

□ 空间复杂度： $O(n^2)$

□ 空间复杂度可以优化至 $O(n)$

■ 假设字符串 ("babad") 为s，它的长度为n

■ dp是大小为 $n * n$ 的二维数组，dp[i][j]表示s[i, j]是否为回文串，存储true、false

■ 如何求出dp[i][j]的值？分2种情况

① 如果s[i, j]的长度 $(j - i + 1) \leq 2$ 时

□ 如果s[i]等于s[j]，那么s[i, j]是回文串，所以dp[i][j] = s[i] == s[j]

② 如果s[i, j]的长度 $(j - i + 1) > 2$ 时

□ 如果s[i + 1, j - 1]是回文串，并且s[i]等于s[j]，那么s[i, j]是回文串

□ 所以dp[i][j] = dp[i + 1, j - 1] && (s[i] == s[j])

扩展中心法

				i	
				↓	
←					→
0	1	2	3	4	5
a	b	b	a	b	a

				i	
				↓	
←					→
0	1	2	3	4	5
a	b	b	a	b	a

- 假设字符串 ("abbaba") 的长度为 n ，那么一共有 $n + (n - 1) == 2n - 1$ 个扩展中心
- 时间复杂度: $O(n^2)$
- 空间复杂度: $O(1)$

基于扩展中心法的优化

i	r							
↓	↓							
0	1	2	3	4	5	6	7	8
b	a	b	b	b	a	b	a	a

■ 算法的核心思想：由连续的相同字符组成的子串作为扩展中心

■ 所以，字符串“babbbabaa”的扩展中心有

□ “b” 、 “a” 、 “bbb” 、 “a” 、 “b” 、 “aa”

■ 核心逻辑

□ 找到右边第一个不等于s[i]的字符，记为位置r，i左边位置记为l

□ r作为下一次的i

□ 由l开始向左、r开始向右扩展，找到最长的回文子串

Manacher (马拉车)

			0		1		2		3		4		5		
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
cs	^	#	a	#	b	#	b	#	a	#	b	#	a	#	\$
m	0	0	1	0	1	4	1	0	3	0	3	0	1	0	0

■ 中间的#字符可以是任意字符，头部的^字符、尾部的\$字符，必须是原字符串中不包含的字符

■ $m[i]$ 的含义

□ 是以 $cs[i]$ 为扩展中心的最大回文子串的长度（不包含#字符）

✓ 最大回文子串在原字符串中的开始索引： $(i - m[i]) \gg 1$

□ 是以 $cs[i]$ 为扩展中心的最大回文子串的右半部分或左半部分的长度（包含#字符）

■ 所以，Manacher算法的关键在于求出m数组

Manacher (马拉车)

			0		1		2		3		4		5		
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
cs	^	#	c	#	b	#	a	#	b	#	c	#	a	#	\$
m	0	0	1	0	1	0	5	0							

↑
l

↑
li

↑
c

↑
i

↑
r

■ 已知

□ 索引 **l**、**li**、**c**、**i**、**r** 的值分别为 1、4、6、8、11

□ $cs[l, r]$ 是以 **c** 为中心的最大回文串

□ **i**、**li** 以 **c** 为中心对称， $m[i]$ 是待求项

□ $m[li] == 1$

□ $i + m[li] < r$

■ 由于回文的对称性，得出结论

□ $m[i] = m[li]$

□ $m[i] == 1$

Manacher (马拉车)

			0		1		2		3		4		5		
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
cs	^	#	a	#	b	#	a	#	b	#	a	#	b	#	\$
m	0	0	1	0	3	0	5	0							

↑
l

↑
li

↑
c

↑
i

↑
r

■ 已知

□ $m[li] == 3$

□ $i + m[li] == r$

■ 结论

□ $m[i]$ 至少是 $m[li]$ ，也就是说，至少是 3

□ 接下来利用扩展中心法以 i 为中心计算出 $m[i]$

■ 当 $i + m[i] > r$ 时，更新 c 、 r

□ $c = i$

□ $r = i + m[i]$

Manacher (马拉车)

			0		1		2		3		4		5		6		7		8		
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
cs	^	#	c	#	b	#	a	#	b	#	c	#	b	#	a	#	b	#	b	#	\$
m	0	0	1	0	1	0	5	0	1	0	7	0	1	0							

↑
l

↑
li

↑
c

↑
i

↑
r

■ 已知

□ $m[li] == 5$

□ $i + m[li] > r$

■ 结论

□ $m[i]$ 至少是 $r - i$, 也就是说, 至少是 3

□ 接下来利用扩展中心法以 i 为中心计算出 $m[i]$

■ 当 $i + m[i] > r$ 时, 更新 c 、 r

□ $c = i$

□ $r = i + m[i]$

Manacher (马拉车)

			0		1		2		3		4		5		
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
cs	^	#	c	#	a	#	b	#	a	#	a	#	a	#	\$
m	0	0	1	0	1	0	3	0	1						

↑
l

↑
c

↑
r
↑
i

■ 当 $i == r$ 时

□ 直接利用扩展中心法以 i 为中心计算出 $m[i]$

■ 当 $i + m[i] > r$ 时, 更新 c 、 r

□ $c = i$

□ $r = i + m[i]$

- 将[面试题47. 礼物的最大价值](#)、[72. 编辑距离](#)、[5. 最长回文子串](#)中的二维数组优化成一维数组
- [1143. 最长公共子序列](#) ([第二季](#)中讲过)
- [53. 最大子序和](#)、[面试题42. 连续子数组的最大和](#) ([第二季](#)中讲过)
- [322. 零钱兑换](#)、[面试题 08.11. 硬币](#) ([第二季](#)中讲过)
- [300. 最长上升子序列](#) ([第二季](#)中讲过)
- [70. 爬楼梯](#) ([第二季](#)中讲过)
- [198. 打家劫舍](#) ([每周一到算法题](#)中讲过)、[213. 打家劫舍 II](#)

- [674. 最长连续递增序列](#)
- [63. 不同路径 II](#)
- [122. 买卖股票的最佳时机 II](#)
- [123. 买卖股票的最佳时机 III](#)
- [188. 买卖股票的最佳时机 IV](#)
- [714. 买卖股票的最佳时机含手续费](#)

- [673. 最长递增子序列的个数](#)
- [1235. 规划兼职工作](#)
- [943. 最短超级串](#)
- [516. 最长回文子序列](#)
- [376. 摆动序列](#)