

算法分析和複雜性理論

干皓丞，2101212850, 信息工程學院

2022 年 3 月 30 日

1 作業目標與章節摘要

1. LeetCode 5. Longest Palindromic Substring 最長回文子串
2. LeetCode 64. Minimum Path Sum 最小路徑和
3. LeetCode 120. Triangle, 三角形最小路徑和

2 作業內容概述

作業可以從 GitHub 下的 `kancheng/kan-cs-report-in-2022` 專案找到，作業程式碼與文件目錄為 `kan-cs-report-in-2022/AATCC/lab-report/`。實際執行的環境與實驗設備為 Google 的 Colab、MacBook Pro (Retina, 15-inch, Mid 2014)、Acer Aspire R7 與 HP Victus (Nvidia GeForce RTX 3060)。

本作業 GitHub 專案為 `kancheng/kan-cs-report-in-2022` 下的 AATCC 的目錄。程式碼可以從 code 目錄下可以找到 *.py 檔案，內容包含上次課堂練習、LeetCode 範例思路整理與作業。

<https://github.com/kancheng/kan-cs-report-in-2022/tree/main/AATCC>



Fig. 1. 作業專案位置

1. LeetCode : <https://leetcode.com/>
2. LeetCode CN : <https://leetcode-cn.com/>
3. OnlineGDB : <https://www.onlinegdb.com/>

LeetCode 的平台部分，CN 的平台有針對簡體中文使用者進行處理，包含中英文切換等功能。OnlineGDB 則可線上進行簡易的環境測試，其程式碼涵蓋 C, C++, C#, Java, Python, JS, Rust, Go。

3 LeetCode 5. Longest Palindromic Substring 最長回文子串

3.1 LeetCode 5. 題目

Given a string s , return the longest palindromic substring in s .

給你一個字符串 s ，找到 s 中最長的回文子串。

Example 1:

```
1 Input: s = "babad"
2 Output: "bab"
3 Explanation: "aba" is also a valid answer.
4 解釋: "aba" 同樣是符合題意的答案。
```

Example 2:

```
1 Input: s = "cbabd"
2 Output: "bb"
```

Constraints:

1. $1 \leq s.length \leq 1000$
 2. s consist of only digits and English letters.
- s 僅由數字和英文字母組成

3.2 LeetCode 56. 思路總結

解法一，動態規劃。定義 $dp[i][j]$ 表示從字符串第 i 個字符到第 j 個字符這一段子串是否是回文串。由回文串的性質可以得知，回文串去掉一頭一尾相同的字符以後，剩下的還是回文串。所以狀態轉移方程是 $dp[i][j] = (s[i] == s[j]) \ \&\& \ ((j-i < 3) \ || \ dp[i+1][j-1])$ ，注意特殊的情況， $j-i == 1$ 的時候，即只有 2 個字符的情況，只需要判斷這 2 個字符是否相同即可。 $j-i == 2$ 的時候，即只有 3 個字符的情況，只需要判斷除去中心以外對稱的 2 個字符是否相等。每次循環動態維護保存最長回文串即可。時間複雜度 $O(n^2)$ ，空間複雜度 $O(n^2)$ 。

解法二，中心擴散法。動態規劃的方法中，我們將任意起始，終止範圍內的字符串都判斷了一遍。其實沒有這個必要，如果不是最長回文串，無需判斷並保存結果。所以動態規劃的方法在空間複雜度上還有優化空間。判斷回文有一個核心問題是找到“軸心”。如果長度是偶數，那麼軸心是中心虛擬的，如果長度是奇數，那麼軸心正好是正中心的那個字母。中心擴散法的思想是枚舉每個軸心的位置。然後做兩次假設，假設最長回文串是偶數，那麼以虛擬中心往 2 邊擴散；假設最長回文串是奇數，那麼以正中心的字符往 2 邊擴散。擴散的過程就是對稱判斷兩邊字符是否相等的過程。這個方法時間複雜度和動態規劃是一樣的，但是空間複雜度降低了。時間複雜度 $O(n^2)$ ，空間複雜度 $O(1)$ 。

解法三，滑動窗口。這個寫法其實就是中心擴散法變了一個寫法。中心擴散是依次枚舉每一個軸心。滑動窗口的方法稍微優化了一點，有些軸心兩邊字符不相等，下次就不會枚舉這些不可能形成回文子串的軸心了。不過這點優化並沒有優化時間複雜度，時間複雜度 $O(n^2)$ ，空間複雜度 $O(1)$ 。

解法四，馬拉車算法。這個算法是本題的最優解，也是最複雜的解法。時間複雜度 $O(n)$ ，空間複雜度 $O(n)$ 。中心擴散法有 2 處有重複判斷，第一處是每次都往兩邊擴散，不同中心擴散多次，實際上有很多重複判斷的字符，能否不重複判斷？第二處，中心能否跳躍選擇，不是每次都枚舉，是否可以利用前一次的信息，跳躍選擇下一中心的中心？馬拉車算法針對重複判斷的問題做了優化，增加了一個輔助數組，將時間複雜度從 $O(n^2)$ 優化到了 $O(n)$ ，空間換了時間，空間複雜度增加到 $O(n)$ 。

<https://books.halfrost.com/leetcode/ChapterFour/0001-0099/0005.Longest-Palindromic-Substring/>

3.3 LeetCode 5. Code 範例

LeetCode 5. Python

```
1 class Solution:
2     def longestPalindrome(self, s: str) -> str:
3         if s == s[::-1]:
4             return s
5         max_len = 1
6         ans = s[0]
7         for i in range(1, len(s)):
8             if i - max_len - 1 >= 0 and s[i - max_len - 1: i + 1] == s[i -
9                 max_len - 1: i + 1][::-1]:
10                ans = s[i - max_len - 1: i + 1]
11                max_len += 2
12            if i - max_len >= 0 and s[i - max_len: i + 1] == s[i - max_len: i +
13                1][::-1]:
14                ans = s[i - max_len: i + 1]
15                max_len += 1
16        return ans
```

3.4 LeetCode 5. 結果

Success Details >

Runtime: 107 ms, faster than 98.70% of Python3 online submissions for Longest Palindromic Substring.

Memory Usage: 14 MB, less than 63.83% of Python3 online submissions for Longest Palindromic Substring.

Fig. 2. LeetCode 5 結果

4 LeetCode 64. Minimum Path Sum 最小路徑和

4.1 LeetCode 64. 題目

Given a $m \times n$ grid filled with non-negative numbers, find a path from top left to bottom right, which minimizes the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

給定一個包含非負整數的 $m \times n$ 網格 grid，請找出一條從左上角到右下角的路徑，使得路徑上的數字總和為最小。

說明：每次只能向下或者向右移動一步。

1	3	1
1	5	1
4	2	1

Fig. 3. Example

Example 1:

```
1 Input: grid = [[1,3,1],[1,5,1],[4,2,1]]
2 Output: 7
3 Explanation: Because the path 1 → 3 → 1 → 1 → 1 minimizes the sum.
4 因為路徑 1→3→1→1→1 的總和最小。
```

Example 2:

```
1 Input: grid = [[1,2,3],[4,5,6]]
2 Output: 12
```

Constraints:

1. $m == \text{grid.length}$
2. $n == \text{grid}[i].\text{length}$
3. $1 \leq m, n \leq 200$
4. $0 \leq \text{grid}[i][j] \leq 100$

4.2 LeetCode 64. 思路總結

1. 給定一個包含非負整數的 $m \times n$ 網格，請找出一條從左上角到右下角的路徑，使得路徑上的數字總和為最小。說明：每次只能向下或者向右移動一步。

2. 在地圖上求出從左上角到右下角的路徑中，數字之和最小的一個，輸出數字和。

3. 這一題最簡單的想法就是用一個二維數組來 DP，當然這是最原始的做法。由於只能往下和往右走，只需要維護 2 列信息就可以了，從左邊推到最右邊即可得到最小的解。

4.3 LeetCode 64. Code 範例

```
1 class Solution(object):
2     def minPathSum(self, grid):
3         #此數組用於記憶化搜索
4         dp = [[0]*len(grid[0]) for i in range(len(grid))]
5         for i in range(len(grid)):
6             for j in range(len(grid[0])):
7                 #在起點的時候
8                 if (i == 0 and j == 0):
9                     dp[i][j] = grid[0][0]
10                #在左邊緣的時候
11                elif (j == 0 and i != 0):
12                    dp[i][j] = dp[i - 1][j] + grid[i][j]
13                #在上邊緣的時候
14                elif (i == 0 and j != 0):
15                    dp[i][j] = dp[i][j-1] + grid[i][j]
16                # 普遍情況下
17                else:
18                    dp[i][j] = min(dp[i - 1][j], dp[i][j - 1]) + grid[i][j]
19        return dp[len(grid) - 1][len(grid[0]) - 1]
```

4.4 LeetCode 64. 結果

Success Details >

Runtime: 172 ms, faster than 31.27% of Python3 online submissions for Minimum Path Sum.

Memory Usage: 15.8 MB, less than 56.54% of Python3 online submissions for Minimum Path Sum.

Fig. 4. LeetCode 64 結果

5 LeetCode 120. Triangle, 三角形最小路徑和

5.1 LeetCode 120. 題目

Given a triangle array, return the minimum path sum from top to bottom.

For each step, you may move to an adjacent number of the row below. More formally, if you are on index i on the current row, you may move to either index i or index $i + 1$ on the next row.

給定一個三角形 `triangle`，找出自頂向下的最小路徑和。

每一步只能移動到下一行中相鄰的結點上。相鄰的結點在這裡指的是下標與上一層結點下標相同或者等於上一層結點下標 + 1 的兩個結點。也就是說，如果正位於當前行的下標 i ，那麼下一步可以移動到下一行的下標 i 或 $i + 1$ 。

Example 1:

```
1 Input: triangle = [[2],[3,4],[6,5,7],[4,1,8,3]]
2 Output: 11
3 Explanation: The triangle looks like:
4     2
5    3 4
6   6 5 7
7  4 1 8 3
8 The minimum path sum from top to bottom is 2 + 3 + 5 + 1 = 11 (underlined above)
9 自頂向下的最小路徑和為 11（即，2 + 3 + 5 + 1 = 11）。
```

Example 2:

```
1 Input: triangle = [[-10]]
2 Output: -10
```

Constraints:

1. $1 \leq \text{triangle.length} \leq 200$
2. $\text{triangle}[0].\text{length} == 1$
3. $\text{triangle}[i].\text{length} == \text{triangle}[i - 1].\text{length} + 1$
4. $-10^4 \leq \text{triangle}[i][j] \leq 10^4$

Follow up: Could you do this using only $O(n)$ extra space, where n is the total number of rows in the triangle? 你可以只使用 $O(n)$ 的額外空間（ n 為三角形的總行數）來解決這個問題嗎？

5.2 LeetCode 120. 思路總結

1. 求出從三角形頂端到底端的最小和。要求最好用 $O(n)$ 的時間複雜度。
2. 這一題最優解是不用輔助空間，直接從下層往上層推。普通解法是用二維數組 DP，稍微優化的解法是一維數組 DP。

5.3 LeetCode 120. Code 範例

```
1 from typing import List
2 class Solution:
3     def minimumTotal(self, triangle: List[List[int]]) -> int:
4         depth = len(triangle)
5         for i in range(-2, -depth-1, -1):
```

```
6         for j in range(depth + 1 + i):
7             triangle[i][j] += min(triangle[i+1][j], triangle[i+1][j+1])
8     return triangle[0][0]
```

5.4 LeetCode 120. 結果

Success [Details >](#)

Runtime: **64 ms**, faster than **97.49%** of Python3 online submissions for Triangle.

Memory Usage: **14.9 MB**, less than **64.98%** of Python3 online submissions for Triangle.

Fig. 5. LeetCode 120 結果