第 2 次作業-作業-HW2

學號：112111207
姓名：陳品霖
作業撰寫時間：180 (mins，包含程式撰寫時間)
最後撰寫文件日期：2024/10/28

本份文件包含以下主題：(至少需下面兩項，若是有多者可以自行新增)

- [✔] 說明內容
- [✔] 個人認為完成作業須具備觀念

回答：

1. 問題如下圖所述，並回答下面問題。

Ans:

a.

```
    def getResult(sentence: str) -> Tuple[List[List[chr]], List[List[chr]]]:
    # 宣告兩個二維陣列，分別為字母1和字母2
    alphabet: List[List[chr]] = [
        ['1', '2', '3', '4', '5', '6', '7', '8', '9', '0'],
        ['Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P'],
        ['A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', ';'],
        ['Z', 'X', 'C', 'V', 'B', 'N', 'M', ',', '.', '/']
    ]
    alphabet2: List[List[chr]] = [
        ['!', '@', '#', '$', '%', '^', '&', '*', '(', ')'],
        ['Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P'],
        ['A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', ':'],
        ['Z', 'X', 'C', 'V', 'B', 'N', 'M', '<', '>', '?']
    ]

    return alphabet, alphabet2
```

b.

```
    def find_position(alphabet: List[List[chr]], alphabet2: List[List[chr]], char:
str) -> Tuple[int, int]:
    # 先查找字元 S 在 alphabet 中的位置
    for i in range(len(alphabet)):
        for j in range(len(alphabet[i])):
```

```
                if alphabet[i][j] == char:
                    return i, j
        # 如果字元 S 不在 alphabet 中，查找 alphabet2
        for i in range(len(alphabet2)):
            for j in range(len(alphabet2[i])):
                if alphabet2[i][j] == char:
                    return i, j
        return -1, -1  # 如果找不到，返回 (-1, -1)

    def process_tests(N: int, tests: List[Tuple[str, int]]) -> None:
        # 取得兩個二維陣列
        alphabet, alphabet2 = getResult("")

        # 定義二維陣列的大小
        rows = len(alphabet)      # 第幾個列表
        cols = len(alphabet[0])   # 第幾個元素

        # 依序處理每一筆測試資料
        for S, K in tests:
            # 找到字元 S 在 alphabet 或 alphabet2 中的位置
            row, col = find_position(alphabet, alphabet2, S)

            # 如果找不到字元 S，就打印 "Not Found"
            if row == -1 and col == -1:
                print("Not Found")
            else:
                # 根據 K 的值來決定輸出的方向
                if K == 1:  # 上
                    new_row = (row - 1) % rows  # 向上移動
                    print(alphabet[new_row][col])  # 在 alphabet 中輸出
                elif K == 2:  # 下
                    new_row = (row + 1) % rows  # 向下移動
                    print(alphabet[new_row][col])  # 在 alphabet 中輸出
                elif K == 3:  # 左
                    new_col = (col - 1) % cols  # 向左移動
                    print(alphabet[row][new_col])  # 在 alphabet 中輸出
                elif K == 4:  # 右
                    new_col = (col + 1) % cols  # 向右移動
                    print(alphabet[row][new_col])  # 在 alphabet 中輸出
            # 每筆資料換行
            print()

    # 主程式，用於測試
    N = int(input("請輸入測試資料的筆數: "))
    tests = []

    # 讀取每筆測試資料
    for _ in range(N):
        S = input("請輸入要檢測的字元: ")
        K = int(input("請輸入方向 K (1: 上, 2: 下, 3: 左, 4: 右): "))
        tests.append((S, K))

    # 呼叫函式處理輸入的測試資料
```

```
process_tests(N, tests)
```

程式結構與流程：

1.定義二維陣列 `alphabet` 和 `alphabet2`：

- `alphabet` 與 `alphabet2` 是兩個二維陣列，模擬鍵盤的字母佈局。`alphabet` 中包含數字和字母，`alphabet2` 中則有特殊符號和字母。
- 兩個陣列的結構類似，主要是用於查找特定字元的位置。

2.`find_position` 函數：

- 接收 `alphabet`, `alphabet2` 和要查找的字元 char。
- 先在 `alphabet` 中查找 `char`，如果找到就返回該字元的行與列位置 `(row, col)`。
- 如果沒找到，則繼續在 `alphabet2` 中查找。
- 如果在兩個陣列中都找不到，返回 `(-1, -1)` 表示「未找到」。

3.`process_tests` 函數：

- 接收測試資料的數量 `N` 和一組包含 `(S, K)` 的測試資料。
- 每一筆測試資料包含一個字元 `S` 和移動方向 `K`。
- 先透過 `getResult` 函數取得 `alphabet` 和 `alphabet2`，並設定陣列的行列大小 `rows` 和 `cols`。
- 然後對每一筆 `(S, K)` 進行處理：
- 利用 `find_position` 函數找出 `S` 在 `alphabet` 或 `alphabet2` 中的位置 `(row, col)`。
- 如果找不到字元 `S`，則輸出 `"Not Found"`。 如果找到位置 `(row, col)`，根據 `K` 的值決定移動方向並輸出：
- `K = 1` (上)：將 `row` 向上移動 `( row - 1 )` ，並使用模數 `%` 避免超出邊界。
- 以此類推

---

2. 給定一個包含 n 個不同數字的數組，這些數字的範圍是從 0 到 n。找出數組中缺失的那一個數字。

Ans:

```python
def find_missing_number(nums):
    n = len(nums)
    expected_sum = n * (n + 1) // 2
    actual_sum = sum(nums)
    return expected_sum - actual_sum
```

```
# 讓使用者輸入數字並轉換為整數列表
user_input = input("請輸入一組數字，數字之間用空格分隔: ")
nums = list(map(int, user_input.split()))

missing_number = find_missing_number(nums)
print("缺少的數字是:", missing_number)
```

程式結構與流程：

`find_missing_number` 函數

1.計算 `n`的值：

- `n`是輸入數字列表的長度，即 `n = len(nums)`。
- 因為陣列應該包含從 `0` 到 `n` 的所有數字，所以這個長度實際上應該是 `n + 1`（因為缺少一個數字）。

2.計算 `expected_sum`（理論總和）：

- 根據等差數列的公式，從 0 到 n 的數字總和為：`\text{expected_sum} = \frac{n \times (n + 1)}{2}`
- 例如，假設 `nums` 是 `[3, 0, 1]`，則 `n = 3`，理論總和應該是`6`

3.計算 `actual_sum`（實際總和）：

- 使用 `Python` 的 `sum`函數計算 `nums` 中所有元素的和，得到 `actual_sum`。
- 例如，對於 `[3, 0, 1]`，實際總和是 `3 + 0 + 1 = 4`

4.計算並返回缺少的數字：

- 缺少的數字就是 `expected_sum - actual_sum`。
- 例如，`6 - 4 = 2`，所以缺少的數字是 `2`。

3. 請回答下面問題：

Ans:

a：

$$ f(n) = 2^n + 1 $$

$$ g(n) = 2^n $$

$$ f(n) <= O(g(n)) $$

$$ 2^n + 1 <= c * g(n) $$

$$ 2^n + 1 <= c * 2^n (2^n 相消) $$

$$ 2^1 <= c (成立) $$

$$ A: 等於O(2^n)

$$

b：

$$ f(n) = 2^2n $$

$$ g(n) = 2^n $$

$$ f(n) <= O(g(n)) $$

$$ 2^2n <= c * g(n) $$

$$ 2^2n <= c * 2^n (2^n 相消) $$

$$ 2^n <= c (不成立) $$

$$ A: 不等於O(2^n) $$

4. 請問以下各函式，在進行呼叫後，請計算(1)執行次數 T(n)，並(2)透過執行次數判斷時間複雜度為何(請用 Big-Oh 進行表示)？

Ans:

a. ：

```
    def calculateTimes(number: int) -> None:
    while number >= 1:                    # n + 1
        counter: int = number              # n
        while counter >= 1:                # (n+1)*n/2
            print(number, counter)         # (n+1)*n/2
            counter = counter - 1          # (n+1)*n/2
        number = number - 1                # n
```

(1)$ T(n) = \frac{3}{2}n^2 + \frac{11}{2}n + 1 $

(2)$T(n) = O(n^2)$

b. ：

```
def calculateTimes(number: int) -> None:
    while number >= 1:                  # floor(log_{2}n) + 2
        print(number)                   # floor(log_{2}n) + 1
        number = number // 2            # floor(log_{2}n) + 1
```

(1) $T(n)=3[(\log x)] + 4$

(2) $T(n) =O(\log_{2}n) $

c. ：

```
def calculateTimes(number: int, size: int) -> None:
    while number >= 1:                      # floor(log_{2}n) + 2
        while size >= 1:                    # (m+1)(floor(log_{2}n) + 1)
            print(number, size)             # m(floor(log_{2}n) + 1)
            size = size - 1                 # m(floor(log_{2}n) + 1)
        number = number // 2                # floor(log_{2}n) + 1
```

(1) $T(n, m) = (3m + 3) \lfloor \log_2 n \rfloor + 3m + 4$

(2) $T(n, m) = O(m \log_2 n)$

d. :

```
#if m=n(最大值)
def calculateTimes(number: int, size: int) -> None:
    while number >= 1:
        #floor(log_{2}n)+2
        while size >= 1:
            print(number, size)
            size = size - 1
        # (n+1)(floor(log_{2}n)+1)
        number = number // 2
        # n(floor(log_{2}n)+1)
        # floor(log_{2}n)+1
```

```
#if m=n/2(最小值)
def calculateTimes(number: int, size: int) -> None:
    while number >= 1:
        # floor(log_{2}n)+2
        while size >= 1:
            print(number, size)
            size = size - 1
        # (n/2+1)(floor(log_{2}n)+1)
        number = number // 2
        # n/2(floor(log_{2}n)+1)
        # floor(log_{2}n)+1
```

(1) $((3n + 3) \lfloor (\log_{2}n) \rfloor + 3n + 4 \geq T(n) \geq \left(\frac{3n}{2} + 3\right) \lfloor (\log_{2}n) \rfloor + \frac{3n}{2} + 4)$

(2) $(T(n) = O(n \log n))$

個人認為完成作業須具備觀念

1. 二維陣列的基本結構：

- 定義和用途：二維陣列是「列表的列表」，可以用來存放具有行列結構的資料。每一個元素都可以用兩個索引值（行和列）來訪問。
- 存取與操作：使用 `array[row][col]` 的方式來存取特定元素。例如 `alphabet[0][1]` 表示二維陣列 `alphabet`中第一行的第二個元素。
- 初始化方式：你可以直接指定數值來初始化，或是使用巢狀迴圈來動態生成。

2. 迴圈與時間複雜度分析：

- 瞭解如何分析迴圈的執行次數（特別是巢狀迴圈的情況），以及如何估計每次迴圈運行所需的步驟數。
- 常見的時間複雜度，例如 $O(n)$、$O(\log n)$、$O(n\log n)$ 等的計算。

3. 對數時間複雜度 $O(\log n)$：

- 瞭解對數複雜度的概念，特別是二元對數 $log_{2}(n)$，以及它在每次將問題大小減半時的應用。

4. 大 O 表示法：

- 理解大 O 表示法的意義和如何使用它來表示演算法的時間複雜度的上界。
- 瞭解如何將不同的項目合併成一個簡單的表示，忽略常數項和低階項。

5. 巢狀迴圈與分治法：

- 當問題涉及巢狀迴圈時，應了解如何計算其疊加的時間複雜度。
- 若涉及二分法或其他分治方法，則應具備一些基本的分治法知識，以便分析執行時間。