LeetCode - Easy

喚起我長蜘蛛網已久的腦袋

- 0009 - Palindrome Number (回文數)
- 0013 - Roman to Integer
- 0014 - Longest Common Prefix
- 0020 - Valid Parentheses
- 0021 - Merge Two Sorted Lists
- 0026 - Remove Duplicates from Sorted Array
- 0027 - Remove Element
- 0028 - Find the Index of the First Occurrence in a String
- 0035 - Search Insert Position
- 0058 - Length of Last Word
- 0066 - Plus One
- 0088 - Merge Sorted Array

0009 - Palindrome Number (回

大意說明

簡單來說就是找到類似於 121 的回文數（就是往前讀往後讀都是 121，英文會長得類似是 abcba 這種東東）

自己的思考過程（逆向字串解）

有幾種想法，一開始先想到透過轉成字串陣列，然後用迴圈做起頭和尾巴的不斷比較，迴圈範圍為字串陣列長度 / 2。

但很顯然地這個會先碰到不同語言對於除法小數的問題，導致無法過關。

例如以下 Python Code 就會卡到 / 2 的問題:

```
class Solution:
    def isPalindrome(self, x: int) -> bool:
        if x < 0:
            return False

        number_str = str(x)
        result = False
        for i in range(0, int(len(number_str) / 2)): # 如果遇到 0 <= x <= 9, len / 2 會是 0
            if number_str[i] == number_str[len(number_str) - 1 - i]:
                result = True
            else:
                result = False
        return result
```

所以後來改開新的 list，倒轉加入之後再把 list 合併成一個字串，就過了:

```
class Solution:
    def isPalindrome(self, x: int) -> bool:
        if x < 0:
            return False
```

```
        number_str = str(x)
        reverse_num = []
        for i in range(0, len(number_str), 1):
            reverse_num.append(number_str[len(number_str) - 1 - i])

        return "".join(reverse_num) == number_str
```

但如果要逆向字串的話, Python 對於字串有內建方法 - Slice:

```
class Solution:
    def isPalindrome(self, x: int) -> bool:
        num_str = str(x)
        reversed_num = num_str[::-1] # Reverse slice
        return reversed_num == num_str
```

Java 版本一開始想到的比較麻煩, 轉成 String 之後, 又要轉成 Char Array 之後才能做逆向加入, 且還要開 StringBuilder (不開也可以但就很耗能):

```
class Solution {
    public boolean isPalindrome(int x) {
        String numStr = String.valueOf(x);
        char[] numChrArray = numStr.toCharArray();
        StringBuilder builder = new StringBuilder();

        for(int i = 0;i < numChrArray.length; i++){
            builder.append(numChrArray[numChrArray.length - 1 - i]);
        }

        return numStr.equals(builder.toString());
    }
}
```

[`String.charAt()`, 於是可以少開一個 Char Array:

```
class Solution {
    public boolean isPalindrome(int x) {
        String numStr = String.valueOf(x);
        StringBuilder reverseNum = new StringBuilder();

        for(int i = 0;i < numStr.length(); i++){
            reverseNum.append(numStr.charAt(numStr.length() - 1 - i));
```

```
        }

        return numStr.equals(reverseNum.toString());
    }
}
```

有效率的解法 ： 數學解

不要做字串轉換，直接做數學計算

除法模數解

- 先判斷是不是小於 0 （因為負數不可能是回文數）
- 跑迴圈
- 用模數取得個位數 (% 10)
- 把個位數加入到另外一個變數當中
- 跑下一次迴圈
- 前面步驟一樣，用模數取得個位數，但是在加入到另外一個變數之前，另外一個變數要先進位 (x 10)
- 跑下一次迴圈
- 迴圈結束條件：當所有的位數都取完了就結束
- 另外一個變數跟原本的數字去比較

相關程式碼如下，以下範例為 Java:

```
class Solution {
    public boolean isPalindrome(int x) {
        if(x < 0){
            return false;
        }

        long temp = x;
        long reversed  = 0;

        while(temp != 0){
            long y = temp % 10;
            reversed = reversed * 10 + y;
            temp /= 10;
        }
```

```
        return x == reversed;
    }
}
```

[illegible]

[illegible]

- [illegible] 10 [illegible] 10 [illegible] 0 [illegible]
- [illegible]
- [illegible] 10 [illegible]

[illegible]

```
class Solution {
    public boolean isPalindrome(int x) {
        if((x < 0) || (x != 0 && x % 10 == 0)){ // [illegible] x [illegible] 0 [illegible] 10 [illegible] 0 [illegible]
            return false;
        }
        long reversed  = 0;

        while(x > reversed){ // [illegible] x [illegible] 10 [illegible] reversed[illegible] loop
            long y = x % 10;
            reversed = reversed * 10 + y;
            x /= 10;
        }

        return (x == reversed) || (x == reversed / 10); // reversed / 10 [illegible]
    }
}
```

0013 - Roman to Integer

大意說明

題幹給了羅馬數字的轉換規則:

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

還有這一段

- `I` can be placed before `V` (5) and `X` (10) to make 4 and 9.
- `X` can be placed before `L` (50) and `C` (100) to make 40 and 90.
- `C` can be placed before `D` (500) and `M` (1000) to make 400 and 900.

思考過程

- 讓程式循序讀字串裡面的字元，然後根據當下字元，以及前一個字元代表的數字去做處理
  - 如果前一個字元沒有遇到 `I`、`X`、`C`，那就照常加入沒差
  - 如果前一個字元有遇到上述字元，因為前面的字元已經加入到 `total` 當中，那只需要把剩下沒加入的數字加入進去，就代表上面的特殊轉換規則了

所以自己的程式碼就長成這樣:

```
class Solution(object):
    def romanToInt(self, s):
        """
        :type s: str
        :rtype: int
        """
        total = 0
        formar_char = ''
```

```
    for char in s:
        if char == 'I':
            total += 1
        elif char == 'V':
            if formar_char == 'I':
                total += 3 # 不是 4 喔, 因為前面的 I 已經加過了 XD, 底下其他也同理
            else:
                total += 5
        elif char == 'X':
            if formar_char == 'I':
                total += 8
            else:
                total += 10
        elif char == 'L':
            if formar_char == 'X':
                total += 30
            else:
                total += 50
        elif char == 'C':
            if formar_char == 'X':
                total += 80
            else:
                total += 100
        elif char == 'D':
            if formar_char == 'C':
                total += 300
            else:
                total += 500
        elif char == 'M':
            if formar_char == 'C':
                total += 800
            else:
                total += 1000
        formar_char = char
    return total
```

更漂亮的解法

一堆 if-else 搞得我好像在寫 yandev...XD

建立一組無序 Map，然後用 Map 下去跑會比較快，然後可以不用 iterator 的方式寫，用傳統 for 寫法可以少存一個變數：

```
class Solution(object):
    def romanToInt(self, s):
        """
        :type s: str
        :rtype: int
        """
        total = 0

        roman_map = {
            'I': 1,
            'V': 5,
            'X': 10,
            'L': 50,
            'C': 100,
            'D': 500,
            'M': 1000
        }

        for i in range(0, len(s)):
            if i != 0 and roman_map[s[i - 1]] < roman_map[s[i]]:
                # 要減兩次的原因是，根據羅馬數字規則，若與後面的大數合併來看的話，前面已經加入的結果要扣除
                total = total + roman_map[s[i]] - (roman_map[s[i - 1]] * 2)
            else:
                total += roman_map[s[i]]

        return total
```

我還看到更白爛的，用 str.replace() 把特殊羅馬數字規則直接替換...我喜歡（但我應該不會先想到這個）

以下則是 Java 版本的程式碼：

```
class Solution {
    public int romanToInt(String s) {
        HashMap<Character, Integer> romanMap = new HashMap<>();
        romanMap.put('I', 1);
        romanMap.put('V', 5);
        romanMap.put('X', 10);
        romanMap.put('L', 50);
```

```
        romanMap.put('C', 100);
        romanMap.put('D', 500);
        romanMap.put('M', 1000);

        int total = 0;
        for(int count = 0; count < s.length(); count++){
            if(count != 0 && romanMap.get(s.charAt(count)) > romanMap.get(s.charAt(count - 1))){
                total -= romanMap.get(s.charAt(count - 1)) * 2;
            }

            total += romanMap.get(s.charAt(count));
        }

        return total;
    }
}
```

0014 - Longest Common Prefix

大意說明

找出一組字串陣列中，相同字串的部份。

[["flower","flow","flight"]] [fl] ["dog","racecar","car"] 的結果是空字串（沒有相同的部份）

解題思路

想不到，所以參考了答案

首先先做字串 list 的排序，原因是字串經過排序之後，一定是由各字串的字母一路排上去，所以最不一樣的肯定在最後面，這樣就可以把比較範圍縮小（也比較不會不知所措）

接著就是第一個字串跟最後一個字串比較，然後用長度最小的字串長度當作迴圈結束條件之一

如果比到一半發現字元不同，直接跳掉。否則就繼續加入字串，直到結束條件達成為止。

程式碼如下:

```
class Solution(object):
    def longestCommonPrefix(self, strs):
        """
        :type strs: List[str]
        :rtype: str
        """

        result = ""

        # 依照字串字母順序排序，最不一樣的一定在最後面
        sorted_list = sorted(strs)

        first = sorted_list[0]
```

```
        last = sorted_list[-1]

        for i in range(0, min(len(first), len(last))):
            if first[i] != last[i]:
                return result
            result += first[i]

        return result
```

□□□□□□□□ `sorted()` □□□Python `Array.sort()` □

□□□ Java code:

```
class Solution {
    public String longestCommonPrefix(String[] strs) {
        ArrayList<String> arraylist = new ArrayList<>(Arrays.asList(strs));
        arraylist.sort(Comparator.naturalOrder());

        StringBuilder samePart = new StringBuilder();

        String first = arraylist.getFirst();
        String last = arraylist.getLast();
        int count = 0;

        while(count < Math.min(first.length(), last.length())){
            if(first.charAt(count) == last.charAt(count)){
                samePart.append(first.charAt(count));
            } else {
                return samePart.toString();
            }
            count++;
        }

        return samePart.toString();
    }
}
```

0020 - Valid Parentheses

大意描述

`'('`、`')'`、`'{'`, `'}'`, `'['` and `']'` ）沒了

[`"()[]{}"` 但實際上還有類似 "{()}" 這種東西，所以不要偷懶 :Kappa:

個人想法

直接用 Stack 的觀念，每次都把暫存在 Stack 內的東西拿出來比較比較，沒有匹配就兩兩丟 Stack，符合就丟掉，直到 Stack 為空代表語法正確:

```
class Solution(object):
    def isValid(self, s):
        """
        :type s: str
        :rtype: bool
        """
        if len(s) < 2:
            return False

        barket_stack = []

        for ch in s:
            if len(barket_stack) == 0:
                barket_stack.append(ch)
                continue

            temp = barket_stack.pop()

            if temp == '(' and ch == ')':
                continue
            if temp == '[' and ch == ']':
                continue
```

```
            if temp == '{' and ch == '}':
                continue
            barket_stack.append(temp)
            barket_stack.append(ch)

        return len(barket_stack) == 0
```

看網路上的解法好像差不多也是 `list[-1]` 的寫法還沒寫以優化了例如以 Python 來說真要拿出來看的話，用 slice 就好(

0021 - Merge Two Sorted Lists

大意

就是把已經排序的兩個 LinkedList 合併到一個 LinkedList, 且要保持順序

實際作法

由於自己做出來都是 Timeout, 應該是哪裡做錯了, 以下附上正確的 Code:

```
# Definition for singly-linked list.
class ListNode(object):
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class Solution(object):
    def mergeTwoLists(self, list1, list2):
        """
        :type list1: Optional[ListNode]
        :type list2: Optional[ListNode]
        :rtype: Optional[ListNode]
        """

        head = ListNode()
        current = head

        while list1 != None and list2 != None:
            if list1.val <= list2.val:
                current.next = list1
                temp = list1.next
```

```
                current = list1
                list1 = temp
            else:
                current.next = list2
                temp = list2.next
                current = list2
                list2 = temp

        if list1 != None:
            current.next = list1
        if list2 != None:
            current.next = list2

        return head.next
```

建立一個新的 LIstNode 當作假的起頭（真正的起頭在假起頭的下一個），然後去比較兩個當前 ListNode 代表的數字

- 如果 list1 的數字比較小，或等於 list2，則新 list 的 next 必須是 list1
- 然後原本 list1 換成 list1 的下一個，也就是 list1.next
- 否則，新 list 的 next 必須為 list2，然後原本 list2 換成 list2.next
- 以此類推，直到 list1 或 list2 其中一個的值是 None (null)
- 但是 list1 或 list2 無論剩餘 LinkedList 的 Node 數量有多少，只要其中一個不是 None (null)，就直接接到新 ListNode 的最後就好（畢竟都排序過了）

然後回傳假的起頭的下一個，就是排序好的 LinkedList!

0026 - Remove Duplicates from Sorted Array

[illegible]

[illegible] `K` [illegible]

[illegible] case:

```
Input: nums = [0,0,1,1,1,2,2,3,3,4]
Output: 5, nums = [0,1,2,3,4,_,_,_,_,_]
Explanation:
 Your function should return k = 5, with the first five elements of nums being 0, 1, 2, 3, and 4 respectively.
It does not matter what you leave beyond the returned k (hence they are underscores).
```

[illegible]

[illegible] dupe [illegible] K [illegible] count [illegible]

[illegible]Java[illegible]

```
class Solution {
    public int removeDuplicates(int[] nums) {
        int dupeCount = 0;
        int count = 0;

        // [illegible] dupe [illegible]
        while(count < nums.length - 1 - dupeCount){
            if(nums[count] == nums[count + 1]){
                // [illegible]dupe [illegible] + 1
                for(int j = count; j < nums.length - 1 - dupeCount; j++){
                    nums[j] = nums[j + 1];
                }
                dupeCount += 1;
```

```
            } else {
                count += 1;
            }
        }

        // 之前結束邊界扣 1 是為了避免 out-of-bound
        // 所以要再 + 1 才能表示原始數字
        return count + 1;
    }
}
```

不過這效率滿差的（會卡在把陣列移動到前面的狀況? 或是兩層迴圈的問題...），不是 LeetCode 前段的執行時間。

後來看了 LeetCode 的討論，有一個只要一個迴圈的思考方式，就是計算 unique index，根據當前位置與前一個位置比較，如果遇到獨特的數，就把他複寫到當前的 unique index:

```
class Solution {
    public int removeDuplicates(int[] nums) {
        int uniqueIndex = 1;

        for(int i = 1; i < nums.length; i++){
            // 如果遇到不重複，把他複寫到 uniqueIndex 所在的位置
            if(nums[i] != nums[i - 1]){
                nums[uniqueIndex] = nums[i];
                uniqueIndex++;
            }
        }

        return uniqueIndex;
    }
}
```

這個時間複雜度就會減少很多了。

有時候可以做反向思考，而不是執著於題幹的描述，應該會對優化算法很有幫助。

0027 - Remove Element

大意

移除陣列裡的特定數字，回傳剩餘數量的整數。

輸出不看排序，只要內容對就好，例如以下題幹:

> **Input:** nums = [0,1,2,2,3,0,4,2], val = 2
> **Output:** 5, nums = [0,1,4,0,3,_,_,_]
> **Explanation:** Your function should return k = 5, with the first five elements of nums containing 0, 0, 1, 3, and 4.
> Note that the five elements can be returned in any order.
> It does not matter what you leave beyond the returned k (hence they are underscores).

詳細說明

因為自己卡關卡太久了，就直接看解答。

[0026](#) 思考不解題策略就會跟

```
class Solution {
    public int removeElement(int[] nums, int val) {
        int nonTargetIndex = 0;

        for(int i = 0; i < nums.length; i++){
            if(nums[i] != val){
                nums[nonTargetIndex] = nums[i];
                nonTargetIndex += 1;
            }
        }

        return nonTargetIndex;
    }
}
```

- 設定一個紀錄非移除目標的 Target Index

- 迭代目前的整數陣列
- 如果目前迭代整數不等於要移除的目標，將它複製至非移除目標的 Target Index, Target Index 移動到下一個位置
- 重複此過程直到迭代結束，回傳非移除目標的值

0028 - Find the Index of the First Occurrence in a String

大意

找出目標字串片段第一次出現的起始 index

例如 mississippi ，目標字串為 issip, 那 index 要回傳為 4

個人思考

因為經常 Time out, 我不知道是哪裡出了問題。

最初是這樣寫的，但是在測資上爆掉了:

```
class Solution {
    public int strStr(String haystack, String needle) {
        int sameCount = 0;
        int lastSuccess = 0;
        int count = 0;

        while(count < haystack.length()){
            if(sameCount == needle.length()){
                return count - sameCount;
            }

            if(haystack.charAt(count) != needle.charAt(sameCount)){
                sameCount = 0;
                if(lastSuccess != 0){
                    count = lastSuccess;
                    continue;
                }
            } else {
```

```
                sameCount++;
                lastSuccess = count;
            }
            count++;
        }

        if(sameCount == needle.length() - 1){
            return haystack.length() - sameCount;
        } else {
            return -1;
        }
    }
}
```

後來看了一下解答，加上理解解題思路之後，就修改成這樣：

```
class Solution {
    public int strStr(String haystack, String needle) {
        int index = 0;

        // 一定要先排除長度小於比較子字串的狀況
        if(haystack.length() < needle.length()){
            return -1;
        }

        // 每換一個起頭就暴力跑完子字串
        // 迴圈結束條件：如果子字串長度 <= 要比較字串長度，那就永遠都不會超過
        for(index = 0;index <= haystack.length() - needle.length(); index++){
            int subCount = 0;

            // 暴力迴圈跑完子字串，發現不同就跳迴圈
            for(subCount = 0; subCount < needle.length(); subCount++){
                if(haystack.charAt(index + subCount) != needle.charAt(subCount)){
                    break;
                }
            }

            // 只有當子字串 Count 長度等於子字串實際長度時，代表找到子字串了，回傳起始 index
            if(subCount == needle.length()){
                return index;
```

```
            }
        }

        // 如果都跑完迴圈還是找不到，那就是真的找不到
        return -1;
    }
}
```

0035 - Search Insert Position

大意

搜尋已經排列好的陣列，找出目標數字，並回傳該數字所在的 Index

如果找不到目標數字，就回傳該插入在哪邊的 Index。

以下是 LeetCode 的三個範例:

Example 1:

Input: nums = [1,3,5,6], target = 5
Output: 2

Example 2:

Input: nums = [1,3,5,6], target = 2
Output: 1

Example 3:

Input: nums = [1,3,5,6], target = 7
Output: 4

個人想法

使用 Binary Search 就可以滿足要求。

不過當所有數字都找不到時，要再判斷一次中位數跟目標數哪個數字比較大，再依照狀況回傳中位數 Index + 1 還是不用 +1:

```
class Solution {
    public int searchInsert(int[] nums, int target) {
        // Binary search
        int left = 0;
        int right = nums.length - 1;
        int mid = 0;
```

```
        while(left <= right){
            mid = (left + right) / 2;
            if(nums[mid] > target){
                // 比 mid 要再往前一格 (因為 target 比較小)
                right = mid - 1;
            } else if (nums[mid] < target){
                // 比 mid 要再往後一格 (因為 target 比較大)
                left = mid + 1;
            } else {
                return mid;
            }
        }

        // 跳出迴圈，代表還找不到結果
        // 再比較一下目前的中位數，如果比中位數還大，就代表要插入到他後面的位置
        // 否則就是插入中位數的當前位置
        if(nums[mid] < target){
            return mid + 1;
        } else {
            return mid;
        }
    }
}
```

0058 - Length of Last Word

大意

有一串以空格隔開的字句，求最後一個單字的長度。

例如：

Input: s = "   fly me   to   the moon "
Output: 4
Explanation: The last word is "moon" with length 4.

個人想法

從 index = 1 開始往前看，往前是確認前面是不是空格，而當前 index 卻不是空格，那這樣計數器需要先 reset。

否則，看到不是空格就一路加下去，直到跳出迴圈為止。

當然，長度小於 2 的字串要先排除掉。

```
class Solution {
    public int lengthOfLastWord(String s) {
        int count = 1;
        int index = 1;
        int spaceCount = 0;

        if(s.length() < 1){
            return 0;
        }

        while(index < s.length()){
            if(s.charAt(index - 1) == ' ' && s.charAt(index) != ' '){
                count = 0;
            }

            if(s.charAt(index) != ' '){
```

```
            count++;
        }

        index++;
    }

    return count;
  }
}
```

0066 - Plus One

大意

有一個整數陣列，求裡面數值 +1 後的狀況

例如:

> **Input:** digits = [1,2,3]
> **Output:** [1,2,4]
> **Explanation:** The array represents the integer 123.
> Incrementing by one gives 123 + 1 = 124.
> Thus, the result should be [1,2,4].

或是

> **Input:** digits = [9]
> **Output:** [1,0]
> **Explanation:** The array represents the integer 9.
> Incrementing by one gives 9 + 1 = 10.
> Thus, the result should be [1,0].

總之需要考慮到進位的狀況。

個人想法

說真的沒很難，但會卡在陣列操作上:

- 把個位數拿出來 +1 看看，如果小於 10 那就放心 +1 後返回原本的陣列
- 如果有進位問題，那就要考量:
  - 多位數問題
  - 比目前的陣列位數還要多一位的狀況 -> 開新陣列，把內容通通丟進去

因為給的是原生型態，`new int[size + 1]`，的方法開新陣列，然後把原本的數字通通倒過去最快:

```
class Solution {
    public int[] plusOne(int[] digits) {
        int lastIndex = digits.length - 1;
        int lastNum = digits[lastIndex];
```

```
        lastNum += 1;

        if(lastNum < 10){
            digits[lastIndex] = lastNum;
            return digits;
        } else {
            digits[lastIndex] = 0;
            int carry = 1;

            for(int i = lastIndex - 1; i >= 0; i--){
                lastNum = digits[i] + carry;

                if(lastNum < 10){
                    digits[i] = lastNum;
                    carry = 0;
                    break;
                } else {
                    digits[i] = 0;
                }
            }

            if(carry > 0){
                int[] newArr = new int[digits.length + 1];
                newArr[0] = carry;

                for(int i = 1; i < newArr.length; i++){
                    newArr[i] = digits[i - 1];
                }

                return newArr;
            } else {
                return digits;
            }
        }
    }
}
```

0088 - Merge Sorted Array

大意來說

其實還滿簡單的...如果你排序演算法都沒有忘記的話

直接看輸出:

> **Input:** nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3
> **Output:** [1,2,2,3,5,6]
> **Explanation:** The arrays we are merging are [1,2,3] and [2,5,6].
> The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.

就是通通要把東西合併到 nums1 然後做 sort 就好

相關 Code

因為連最簡單的泡沫排序都忘記了，所以其實花很久時間處理 XD

```
class Solution(object):
    def merge(self, nums1, m, nums2, n):
        """
        :type nums1: List[int]
        :type m: int
        :type nums2: List[int]
        :type n: int
        :rtype: None Do not return anything, modify nums1 in-place instead.
        """

        for i in range(m, m+n):
            nums1[i] = nums2[m - i]

        # Sort: Bubble sort
        temp_index = 0
        for i in range(0, m+n - 1):
            for j in range(0, m+n - 1 - i):
```

```
            if nums1[j] > nums1[j + 1]:
                temp = nums1[j]
                nums1[j] = nums1[j + 1]
                nums1[j + 1] = temp
```

是時候該來去惡補排序演算法了...。