目錄

7. 二元樹Binary Tree 1.7
    1. LeetCode 104. Maximum Depth of Binary Tree 1.7.1
    2. LeetCode 226. Invert Binary Tree 1.7.2
    3. LeetCode 100. Same Tree 1.7.3
    4. LeetCode 235. Lowest Common Ancestor of a Binary Search 1.7.4
    5. LeetCode 102. Binary Tree Level Order Traversal 1.7.5
    6. LeetCode 107. Binary Tree Level Order Traversal II 1.7.6
    7. LeetCode 101. Symmetric Tree 1.7.7
    8. LeetCode 110. Balanced Binary Tree 1.7.8
    9. LeetCode 112. Path Sum 1.7.9
    10. LeetCode 111. Minimum Depth of Binary Tree 1.7.10
    11. LeetCode 257. Binary Tree Paths 1.7.11
8. 連結串列LinkedList 1.8
    1. LeetCode 206. Reverse Linked List 1.8.1
    2. LeetCode 237. Delete Node in a Linked List 1.8.2
    3. LeetCode 83. Remove Duplicates from Sorted List 1.8.3
    4. LeetCode 141. Linked List Cycle 1.8.4
    5. LeetCode 21. Merge Two Sorted Lists 1.8.5
    6. LeetCode 24. Swap Nodes in Pairs 1.8.6
    7. LeetCode 160. Intersection of Two Linked Lists 1.8.7
    8. LeetCode 19. Remove Nth Node From End of List 1.8.8
    9. LeetCode 234. Palindrome Linked List 1.8.9
    10. LeetCode 203. Remove Linked List Elements 1.8.10
    11. LeetCode 2. Add Two Numbers 1.8.11
9. 動態歸劃Optimization 1.9
    1. LeetCode 292. Nim Game 1.9.1
    2. LeetCode 70. Climbing Stairs 1.9.2
    3. LeetCode 121. Best Time to Buy and Sell Stock 1.9.3
    4. LeetCode 198. House Robber 1.9.4
    5. LeetCode 374. Guess Number Higher or Lower 1.9.5
    6. LeetCode 278. First Bad Version 1.9.6
10. 其他Others 1.10
    1. LeetCode 371. Sum of Two Integers 1.10.1
    2. LeetCode 191. Number of 1 Bits 1.10.2
    3. LeetCode 232. Implement Queue using Stacks 1.10.3
    4. LeetCode 36. Valid Sudoku 1.10.4
    5. LeetCode 299. Bulls and Cows 1.10.5
    6. LeetCode 225. Implement Stack using Queues 1.10.6
    7. LeetCode 190. Reverse Bits 1.10.7
    8. LeetCode 303. Range Sum Query - Immutable 1.10.8
    9. LeetCode 155. Min Stack 1.10.9

README

LeetCode with Javascript

by YanQY(驗) at 2016

開始閱讀

github連結

關於**LeetCode**

LeetCode是一個線上的網站，提供IT人員面試時常常會遇到的現場coding題目， 這些題目其實是可以透過練習來熟悉理解，在練習的同時，IT人員也可以增進自己coding的能力。LeetCode除了將這些題目 搬上到網頁上讓使用者可以選擇不同的程式語言進行線上coding之外，也有討論區可以分享或是看其他人解題的思路想法。

題目有分爲，Easy，Medium，Hard三個等級，每題後面還有目前線上使用者提交程式碼的通過率，一般來說，同樣等級的題目， 通過率越高的題目爲越簡單，這也是我建議的解題順序，先從Easy，通過率最高的一題開始。

版權許可**(License**）

關於這份文件

一開始練習LeetCode的時候，深深感受到自己coding相關的基礎實在太差，演算法，資料結構等等都還要再加強。 解題的時候往往想了很久還要去參考別人的寫法才能順利通過測試，但常常我是用大量的試誤法Trail & Error來拼湊出正確的答案， 對於題目應該要怎麼解沒有一個比較確切的想法，這份文件主要是整理自己解題的方向以及解題過程中的一些心得。

目前只包含LeetCode Easy的題目，而且只有免費的部分，目前還沒有打算去解付費的題目，預設排序是通過率從高到低， 主要目的是盡量清楚明白的解釋爲什麼要這樣寫，因此程式碼一定不會很簡潔，執行的速度也不是最快，這點請多多包涵， 有發現錯誤請到這邊 留言，感激不盡。

作者

YanQY

mail: skyyen999@gmail.com

github: https://github.com/skyyen999

通過率排序

- 難易度排序

難易度排序

#	題目	通過率
344	Reverse String	59%
292	Nim Game	54%
371	Sum of Two Integers	52%
258	Add Digits	49%
104	Maximum Depth of Binary Tree	49%
226	Invert Binary Tree	47%
283	Move Zeroes	46%
349	Intersection of Two Arrays	44%
237	Delete Node in a Linked List	44%
100	Same Tree	44%
171	Excel Sheet Column Number	43%
242	Valid Anagram	43%
169	Majority Element	43%
217	Contains Duplicate	42%
350	Intersection of Two Arrays II	42%
13	Roman to Integer	41%
206	Reverse Linked List	41%
231	Power of Two	38%
326	Power of Three	38%
191	Number of 1 Bits	38%
202	Happy Number	38%
263	Ugly Number	38%
83	Remove Duplicates from Sorted List	38%
235	Lowest Common Ancestor of a Binary Search Tree	38%
70	Climbing Stairs	37%
121	Best Time to Buy and Sell Stock	37%
141	Linked List Cycle	36%
21	Merge Two Sorted Lists	36%
345	Reverse Vowels of a String	36%
24	Swap Nodes in Pairs	36%
198	House Robber	36%
342	Power of Four	36%
107	Binary Tree Level Order Traversal II	35%
27	Remove Element	35%
101	Symmetric Tree	35%

通過率排序

66	Plus One	35%
110	Balanced Binary Tree	35%
118	Pascal's Triangle	35%
232	Implement Queue using Stacks	34%
26	Remove Duplicates from Sorted Array	34%
102	Binary Tree Level Order Traversal	34%
172	Factorial Trailing Zeroes	34%
119	Pascal's Triangle II	33%
9	Palindrome Number	33%
36	Valid Sudoku	32%
112	Path Sum	32%
374	Guess Number Higher or Lower	31%
299	Bulls and Cows	31%
111	Minimum Depth of Binary Tree	31%
205	Isomorphic Strings	31%
225	Implement Stack using Queues	31%
223	Rectangle Area	31%
257	Binary Tree Paths	31%
160	Intersection of Two Linked Lists	31%
219	Contains Duplicate II	31%
88	Merge Sorted Array	31%
19	Remove Nth Node From End of List	31%
290	Word Pattern	31%
38	Count and Say	30%
20	Valid Parentheses	30%
58	Length of Last Word	30%
203	Remove Linked List Elements	30%
234	Palindrome Linked List	30%
190	Reverse Bits	29%
14	Longest Common Prefix	29%
67	Add Binary	29%
28	Implement strStr()	26%
303	Range Sum Query - Immutable	26%
204	Count Primes	25%
1	Two Sum	25%
6	ZigZag Conversion	25%
125	Valid Palindrome	24%
7	Reverse Integer	24%
155	Min Stack	24%
278	First Bad Version	23%
168	Excel Sheet Column Title	23%
189	Rotate Array	22%
165	Compare Version Numbers	18%
8	String to Integer (atoi)	14%

通過率排序

題號排序

- 題號排序

題號排序

#	題目	通過率
1	Two Sum	25%
6	ZigZag Conversion	25%
7	Reverse Integer	24%
8	String to Integer (atoi)	14%
9	Palindrome Number	33%
13	Roman to Integer	41%
14	Longest Common Prefix	29%
19	Remove Nth Node From End of List	31%
20	Valid Parentheses	30%
21	Merge Two Sorted Lists	36%
24	Swap Nodes in Pairs	36%
26	Remove Duplicates from Sorted Array	34%
27	Remove Element	35%
28	Implement strStr()	26%
36	Valid Sudoku	32%
38	Count and Say	30%
58	Length of Last Word	30%
66	Plus One	35%
67	Add Binary	29%
70	Climbing Stairs	37%
83	Remove Duplicates from Sorted List	38%
88	Merge Sorted Array	31%
100	Same Tree	44%
101	Symmetric Tree	35%
102	Binary Tree Level Order Traversal	34%
104	Maximum Depth of Binary Tree	49%
107	Binary Tree Level Order Traversal II	35%
110	Balanced Binary Tree	35%
111	Minimum Depth of Binary Tree	31%
112	Path Sum	32%
118	Pascal's Triangle	35%
119	Pascal's Triangle II	33%
121	Best Time to Buy and Sell Stock	37%
125	Valid Palindrome	24%
141	Linked List Cycle	36%

155	Min Stack	24%
160	Intersection of Two Linked Lists	31%
165	Compare Version Numbers	18%
168	Excel Sheet Column Title	23%
169	Majority Element	43%
171	Excel Sheet Column Number	43%
172	Factorial Trailing Zeroes	34%
189	Rotate Array	22%
190	Reverse Bits	29%
191	Number of 1 Bits	38%
198	House Robber	36%
202	Happy Number	38%
203	Remove Linked List Elements	30%
204	Count Primes	25%
205	Isomorphic Strings	31%
206	Reverse Linked List	41%
217	Contains Duplicate	42%
219	Contains Duplicate II	31%
223	Rectangle Area	31%
225	Implement Stack using Queues	31%
226	Invert Binary Tree	47%
231	Power of Two	38%
232	Implement Queue using Stacks	34%
234	Palindrome Linked List	30%
235	Lowest Common Ancestor of a Binary Search Tree	38%
237	Delete Node in a Linked List	44%
242	Valid Anagram	43%
257	Binary Tree Paths	31%
258	Add Digits	49%
263	Ugly Number	38%
278	First Bad Version	23%
283	Move Zeroes	46%
290	Word Pattern	31%
292	Nim Game	54%
299	Bulls and Cows	31%
303	Range Sum Query - Immutable	26%
326	Power of Three	38%
342	Power of Four	36%
344	Reverse String	59%
345	Reverse Vowels of a String	36%
349	Intersection of Two Arrays	44%
350	Intersection of Two Arrays II	42%
371	Sum of Two Integers	52%
374	Guess Number Higher or Lower	31%

題號排序

LeetCode 344. Reverse String

LeetCode 344. Reverse String

題目

Write a function that takes a string as input and returns the string reversed.

Example: Given s = "hello", return "olleh".

翻譯

將一個字串反轉後回傳。

範例： s= "hello", return "olleh"

思路

這題很簡單，應該是所有題目最簡單的一題，如果你解不出來建議不要往下看，回頭在重新學一下怎麼寫程式。

這題的解法是這樣，將字串拆成一個陣列，然後將陣列從後面開始串成一個字串並回傳。
ex. "hello" -> 拆開['h','e','l','l','o'] -> 'o'+'l'+'l'+'e'+'h'

解題

```
var reverseString = function(s) {
    var result = "";        //回傳的字串
    var ary = s.split("");  //字串拆成陣列

    for(var i = ary.length-1 ; i >= 0 ; i--){
        result = result + ary[i];  //從後面串回字串
    }
    return result;
};
```

進階

在陣列內直接將頭尾元素交換，接著交換第二個元素與倒數第二個元素，這樣的做法只要跑一半的迴圈就可以，速度上會快很多。
下面範例因爲切成array來做，所以實際上速度會比較慢。

未交換前 ['h','e','l','l','o']
第1次交換 ['**h**','e','l','l','**o**'] o,h互換 ['**o**','e','l','l','**h**']
第2次交換 ['o','**e**','l','**l**','h'] e,l互換 ['o','**l**','l','**e**','h']

```
/**
 * @param {string} s
 * @return {string}
 */
var reverseString = function(s) {
    var result = "";
    var ary = s.split("");
    for(var i = 0, max = (ary.length-1)/2 ; i < max   ; i++){
        var temp = ary[i];
        ary[i] = ary[ary.length - 1 - i];
        ary[ary.length - 1 - i] = temp;
    }
    return ary.join("");
};
```

LeetCode 242. Valid Anagram

LeetCode 242. Valid Anagram

題目

Given two strings s and t, write a function to determine if t is an anagram of s.

For example,
s = "anagram", t = "nagaram", return true.
s = "rat", t = "car", return false.

Note:
You may assume the string contains only lowercase alphabets.

翻譯

給兩個字串s與t，回傳t是否爲s的重組字

範例：
s = "anagram", t = "nagaram" 回傳true
s = "rat", t = "car" 回傳false

思路

要比較兩個字串裡面的字元是否相同，首先可以判斷長度是否相等，不相等就可以直接判定爲false， 接下來將重新排序後的字串比較是否相等。

解題

```
/**
 * @param {string} s
 * @param {string} t
 * @return {boolean}
 */
var isAnagram = function(s, t) {
    if(s.length != t.length) return false;

    var s = s.split("").sort().join("");
    var t = t.split("").sort().join("");

    return s == t;
```

```
};
```

LeetCode 13. Roman to Integer

LeetCode 13. Roman to Integer

題目

Given a roman numeral, convert it to an integer.

Input is guaranteed to be within the range from 1 to 3999.

翻譯

給一個羅馬數字符號，將之轉爲整數，這個數字一定落在1 到 3999 之間。

範例：
I = 1, IX = 9

思路

使用一個map來儲存羅馬符號跟數字之間的對應關係，在一般的情況下(ex. III, VI)，可以直接將羅馬符號轉換成數字。

不過如果出現IV，XC這種組合，就要另外處理，這種組合的特色是後面的符號會大於前面的符號，因此一次讀兩個羅馬符號來找出這種組合。 一次讀兩個羅馬數字，如果第二個數字(n2)比第一個(n1)大，整數值爲n2-n1，如果是一般的情況n2 <= n1，整數值爲n2+n1。

解題

```
/**
 * @param {string} s
 * @return {number}
 */
var romanToInt = function(s) {
    var map = {
        I:1,
        V:5,
        X:10,
        L:50,
        C:100,
        D:500,
        M:1000,
    }
```

```
    var sum = 0;
    for(var i = 0 ; i < s.length ; i++){
        var v1 = map[s[i]];
        var v2 = map[s[i+1]];
        if(v2 > v1){
            sum = sum + v2 - v1;
            i++;
        }  else {
            sum = sum+v1;
        }
    }
    return sum;
};
```

LeetCode 345. Reverse Vowels of a String

LeetCode 345. Reverse Vowels of a String

題目

Write a function that takes a string as input and reverse only the vowels of a string.

Example 1: Given s = "hello", return "holle".

Example 2: Given s = "leetcode", return "leotcede".

Note: The vowels does not include the letter "y".

翻譯

給一個英文字串，將裡面的母音字母反轉。

範例1:
Given s = "hello", return "holle".

範例2：
Given s = "leetcode", return "leotcede".

注意： y不算在母音字母中。

思路

1. 先將字串中的母音儲存在array
2. 重新搜索字串中的母音，將array理面的字元依相反順序取代字串中的母音

解題

```
/**
 * @param {string} s
 * @return {string}
 */
var reverseVowels = function(s) {

    var vowels = [];  // 儲存找到的母音

    // 找母音
    for(var i = 0 ; i< s.length ; i++){
```

```
        if((/^[aeiou]$/i).test(s[i])){
            vowels.push(s[i]);
        }
    }

    var v = vowels.length - 1;
    // 因爲之前是用array儲存找到的母音，所以把本來的字串也轉成array
    var sAry = s.split("");

    // 用之前找到的array取代母音
    for(var j = 0 ; j < sAry.length ; j++){
        if((/^[aeiou]$/i).test(sAry[j])){
            sAry[j] = vowels[v--];
        }
    }

    return sAry.join("");
};
```

LeetCode 205. Isomorphic Strings

LeetCode 205. Isomorphic Strings

題目

Given two strings s and t, determine if they are isomorphic.

Two strings are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character but a character may map to itself.

For example, Given "egg", "add", return true.

Given "foo", "bar", return false.

Given "paper", "title", return true.

翻譯

給兩個字串s跟t，判斷他們是否是同構字。

如果他們是同構字，表示s裡面每個字元都可以拿來對應t的特定字元。

全部的字元都要依順序被取代，而且s一種字元只會對應t一種字元，也可能對應到與自己相同的字元。

思路

需要兩個map，一個記錄s對t的對應關係，另外一個記錄t對s，如果字元沒在s中出現，加到map，出現過的話就拿出來跟t比對，不一致表示非同構字。

解題

```
/**
 * @param {string} s
 * @param {string} t
 * @return {boolean}
 */
var isIsomorphic = function(s, t) {
    // 使用map來記錄s,t的對應關係
    var mapS = {};
```

```
    var mapT = {};

    for(var i in s){
        var valueS = s[i];
        var valueT = t[i];

        // 如果這個字元還沒出現過，加到mapS中
        if(!mapS[valueS]){
            mapS[valueS] = valueT;
        } else if(mapS[valueS] != valueT) {
            // 如果s字元出現過，比對t的字元使否與mapS中儲存的一樣
            return false;
        }

        if(!mapT[valueT]){
            mapT[valueT] = valueS;
        } else if(mapT[valueT] != valueS) {
            return false;
        }
    }
    // 全部比對成功
    return true;
};
```

LeetCode 290. Word Pattern

LeetCode 290. Word Pattern

題目

Given a pattern and a string str, find if str follows the same pattern.

Here follow means a full match, such that there is a bijection between a letter in pattern and a non-empty word in str.

Examples: pattern = "abba", str = "dog cat cat dog" should return true. pattern = "abba", str = "dog cat cat fish" should return false. pattern = "aaaa", str = "dog cat cat dog" should return false. pattern = "abba", str = "dog dog dog dog" should return false. Notes: You may assume pattern contains only lowercase letters, and str contains lowercase letters separated by a single space.

翻譯

給一個樣版跟一個字串，判斷字串是否有遵循樣版的格式。

這邊的遵循是說字串中的每一個詞都與樣版有一致的對應。

範例：
pattern = "abba", str = "dog cat cat dog" should return true. pattern = "abba", str = "dog cat cat fish" should return false. pattern = "aaaa", str = "dog cat cat dog" should return false. pattern = "abba", str = "dog dog dog dog" should return false.

可以假設樣版中只有小寫字母，字串中每個詞用空白隔開。

思路

1. 這跟205. Isomorphic Strings可以說是95%以上的像
2. 一樣用map儲存出現過的樣版與字串的對應關係，當發現對應不成立，就回傳false

解題

```
/**
 * @param {string} pattern
 * @param {string} str
 * @return {boolean}
 */
var wordPattern = function(pattern, str) {
    var patternMap = {};
```

```
    var strMap = {};
    var ary = str.split(/\s/)

    // 樣版的長度跟字串中單詞的數量對不起來 false
    if(pattern.length != str.split(/\s/).length){
        return false;
    }

    for(var i in pattern){
        var p = pattern[i];
        var s = ary[i];
        // 沒出現過的配對加入map，出現過就進行比對
        if(!patternMap[p]){
            patternMap[p] = s;
        } else if(patternMap[p] != s){
            return false;
        }

        if(!strMap[s]){
            strMap[s] = p;
        }  else if(strMap[s] != p) {
            return false;
        }

    }

    return true;
};
```

LeetCode 38. Count and Say

LeetCode 38. Count and Say

題目

The count-and-say sequence is the sequence of integers beginning as follows: 1, 11, 21, 1211, 111221, ...

1 is read off as "one 1" or 11. 11 is read off as "two 1s" or 21. 21 is read off as "one 2, then one 1" or 1211. Given an integer n, generate the nth sequence.

Note: The sequence of integers will be represented as a string.

翻譯

這是一個算完說出來的序列，序列如下：
1, 11, 21, 1211, 111221, ...

```
1      讀做 1個1， 所以下一個變成 11
11     讀做 2個1， 也就是21
21     讀做 1個2 1個1，得到1211
1211   1個1，1個2，2個1  111221
```

思路

相信這題很多人跟我一樣，看懂題目後就覺得簡單了，直接看下面的code就行。

解題

```
/**
 * @param {number} n
 * @return {string}
 */
var countAndSay = function(n) {
    if(n <= 1) return "1";

    var countSay = '1';

    for(var i = 2 ; i <= n ; i++){
        var num = countSay.charAt(0); //從開頭開始數
        var temp = countSay;
```

```
        var count = 1;

        countSay = ''; // 清空儲存這輪數完的字串


        for(var j = 1 ; j < temp.length; j++){
            // 數字相同，count++
            if(temp.charAt(j) == num){
                count++;
            } else {
                // 數字不同，將目前的count與num加到字串，更新num，新的num從
                countSay += count;
                countSay += num;
                num = temp.charAt(j);
                count = 1;
            }
        }
        countSay += count;
        countSay += num;
    }
    return countSay;

};
```

LeetCode 20. Valid Parentheses

LeetCode 20. Valid Parentheses

題目

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not.

翻譯

給一個只包含'(', ')', '{', '}', '[' , ']'這些括號字元的字串，判斷這些括號是不是合法的。 右括號必須依照正確的順序出現，"()" 與 "()[]{}" 都是合法的，但"(]" 和 "([)]"就不是。

思路

用stack先進後出的特性來解，遇到左括號就放入stack，遇到右括號就取出stack裡面的左括號比對是否合法。

例如說字串 " ([ )] " ， '(' 左括號，放入stack -> stack='(', '['放入sta
')'右括號，取出stack -> '[' ， [)是一個不match的組合，就回傳false

解題

```
/**
 * @param {string} s
 * @return {boolean}
 */
var isValid = function(s) {
    if(!s) return true;

    // 使用stack還儲存左括號
    var stack = [];

    var left = ['(','[','{'];
    var right = [')',']','}'];
    var match = {
        ')':'(',
        ']':'[',
        '}':'{'
```

```
    }

    for(var i in s){
        // 左括號，放入stack
        if(left.indexOf(s[i]) > -1){
            stack.push(s[i]);
        }

        // 右括號，從stack取出左括號判斷是否match
        if(right.indexOf(s[i]) > -1){
            var stackStr = stack.pop();
            if(match[s[i]] != stackStr) {
                return false;
            }
        }
    }

    // 如果左右括號都match的話，stack應該爲空
    return stack.length == 0;
};
```

LeetCode 58. Length of Last Word

LeetCode 58. Length of Last Word

題目

Given a string s consists of upper/lower-case alphabets and empty space characters ' ', return the length of last word in the string.

If the last word does not exist, return 0.

Note: A word is defined as a character sequence consists of non-space characters only.

For example, Given s = "Hello World", return 5.

翻譯

給一個字串s，其中包含大小寫字母與空白' '，回傳最後一個單字的長度，如果沒有最後一個單字，回傳0。

注意：單字的定義是由一串連續中間沒空白的字元所組成。

範例： Given s = "Hello World"，最後一個單字爲world，長度爲5。

思路

1. 用split把字串拆成陣列，例如 "Hello World " -> ['Hello','World', ' ']
2. 取陣列最後一個元素，上面範例取出來的是空字串' '
3. 空字串不是我們要的單字，往前再取到'World'，因此最後一個單字長度爲5

解題

```
/**
 * @param {string} s
 * @return {number}
 */
var lengthOfLastWord = function(s) {
    var ary = s.split(/\s/); //使用regex將字串切開

    if(s.length == 0) return 0;
    if(ary.length == 0) return 0;

    // 從陣列最後面取出單字，如果是空白' '就不是單字，所以繼續往前找
```

```
    while(ary.length > 0){
        var v = ary.pop();
        if(v.length > 0){
            return v.length;
        }

    }

    return 0;
};
```

LeetCode 14. Longest Common Prefix

LeetCode 14. Longest Common Prefix

題目

Write a function to find the longest common prefix string amongst an array of strings.

翻譯

一個陣列中有許多個字串，寫一個function找出這些字串最長的共同字首。

範例：
['abcd','abccc','abdec'] --> 共同字首爲 'ab' 。

思路

1. 比對前兩個字串，從頭開始取出相同的部分爲共同字首
2. 後面的字串只要與目前的共同字首比對即可
3. ['abcd','abccc','abdec'] ，一開始'abcd','abccc'共同字首前3碼'abc'
4. 接下來只要將'abc','abdec'做比對，發現剩下'ab'，也就是最長的共同字首

解題

```
/**
 * @param {string[]} strs
 * @return {string}
 */
var longestCommonPrefix = function(strs) {
    if(strs == null || strs.length == 0) return "";

    // same表示目前發現的共同字首，一開始爲strs[0]
    var same = strs[0];

    // 只需要比對same跟目前字串共同的字元就好
    for(var i = 1 ; i<strs.length ; i++){
        var str = strs[i];

        // 取目前的字串str跟same相等的部分做爲新的same
        var j = 0;
        for(; j < same.length ; j++){
           if(same[j] != str.charAt(j)){
```

```
                break;
            }
        }
        // same與目前字串str前幾位相同，就做爲新的same
        same = same.slice(0,j);
    }

    return same;
};
```

LeetCode 67. Add Binary

LeetCode 67. Add Binary

題目

Given two binary strings, return their sum (also a binary string).

For example, a = "11" b = "1" Return "100".

翻譯

給兩個字元字串，回傳他們的總和(以字元字串回傳)。

範例：
a = '11'
b = '1' return '100'

思路

1. 一開始的想法是將a,b轉爲數字，相加後再轉回bits，程式碼在下面，不過題目給的string長度轉成int時會超過int max，所以這個方法不能用
2. 改用純字串來處理，先處理位數一樣的部分，ex. "111"+"10" ，先處理"11"+"10"
3. 以 "11"+"10"爲例，個位數1+0=1； 十位數1+1=2，因爲bits是二進位，因此2->0，然後儲存進位資訊，這時候要回傳的字串爲'10'
4. "111"最前面的"1"+剛才的進位1 = 2， 2-> 0 ， 會傳字串爲'010'
5. 因爲剛才還有進位，所以要把進位的1加到字串前面，最後回傳'1010'

解題

```
/**
 * @param {string} a
 * @param {string} b
 * @return {string}
 */
var addBinary = function(a, b) {
    // 使用字串來處理
    var sumStr = "";
    var carry = 0;
    var longStr, shortStr;
```

```
    // 判斷a,b哪個字串比較長
    if(a.length > b.length){
        longStr = a;
        shortStr = b;
    } else {
        longStr = b;
        shortStr = a;
    }

    longStr = longStr.split("").reverse().join("");
    shortStr = shortStr.split("").reverse().join("");

    // 將短字串加到長字串裡面，這邊先計算到短字串的位數，ex. "110" + "1111"，
    for(var i = 0 ; i < shortStr.length ; i++) {
        var c = parseInt(shortStr.charAt(i))+parseInt(longStr.charAt
        // 二進位，相加大於1，就要進位
        if(c > 1){
            carry = 1;
            c = c%2;
        } else {
            carry = 0;
        }

        sumStr = c + sumStr;
    }

    // 處理長字串剩下的字串
    for(var j =  shortStr.length  ; j <  longStr.length  ; j ++){
        var c = parseInt(longStr.charAt(j)) + carry;
        if(c > 1){
            carry = 1;
            c = c%2;
        } else {
            carry = 0;
        }
        sumStr = c + sumStr
    }

    // 如果加完後還有進位，要放到字串最前面
    return (carry == 1 ? carry : "") + sumStr;
};
```

失敗的解題

```
var addBinary = function(a, b) {
    var sum = parseInt(a,2)+parseInt(b,2);
    var bitStr = "";

    while(sum > 0){
        bitStr = sum%2 + bitStr;
        sum = Math.floor(sum/2);
    }
```

```
    return bitStr == '' ? '0': bitStr ;
}
```

LeetCode 28. Implement strStr()

LeetCode 28. Implement strStr()

題目

Implement strStr().

Returns the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

翻譯

實做strStr()。

給一個指針needle跟字串堆疊haystack，回傳指針第一次在堆疊中出現的位置index。

範例：
haystack = "abcdede"
needle = "de"，"de"第一次出現的位置爲3

思路

1. 這就是字串搜尋的功能，用api的話，只要用String的indexOf(x)就可以直接得到答案
2. 再來是用截字串的方法，haystack(i,needle.length)如果跟needle相等，回傳i的位置即可
3. 如果不用api的情況下，使用迴圈比對haystack跟needle
4. haystack = "abcbb"，needle = "bb" 爲例
5. 一開始 ***a***bcbb != ***b***b 失敗 ； 往下比對a***b***cbb != ***b***b可是接下來 ab***c***bb != b***b*** 失敗
6. 值到最後ab***bb*** == ***bb***比對成功，回傳haystack現在的位置3
7. 以上寫法效率太差，實際上可以用KMP演算法來增加速度，不過KMP演算法我覺得超出easy的部分，這邊就先跳過不提

解題

```
/**
 * @param {string} haystack
 * @param {string} needle
 * @return {number}
 */
var strStr = function(haystack, needle) {
    return haystack.indexOf(needle);
```

```
}
```

截字串比對

```
var strStr = function(haystack, needle) {
    if(!needle) return 0;
    if(!haystack || needle.length > haystack.length) return -1;

    var i,j;
    for(i = 0 ; i < haystack.length ; i++){
       var str = haystack.substr(i,needle.length);
       if(str == needle){
           return i;
       }
    }

    return -1;
};
```

超出時間的解題

```
var strStr = function(haystack, needle) {
    if(!needle) return 0;
    if(!haystack || needle.length > haystack.length) return -1;

    var i,j;
    for(i = 0 ; i < haystack.length ; i++){
       var index = i;
       j = 0; //比對失敗，將haystack指針移向下一個字元，重新跟needle比對
       while(haystack.charAt(index) == needle.charAt(j) ){
           //比對needle跟haystack，比對成功回傳haystack開始比對的位置
           if(j == needle.length -1){
                return parseInt(i);
           }
           index++; j++;
       }
    }

    return -1;
};
```

LeetCode 6. ZigZag Conversion

LeetCode 6. ZigZag Conversion

題目

The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

```
P   A   H   N
A P L S I I G
Y   I   R
```

And then read line by line: "PAHNAPLSIIGYIR" Write the code that will take a string and make this conversion given a number of rows:

```
string convert(string text, int nRows);
```

convert("PAYPALISHIRING", 3) should return "PAHNAPLSIIGYIR".

翻譯

字串"PAYPALISHIRING"經過Z字轉換後如圖所示，重組後變成"PAHNAPLSIIGYIR"， 寫一個convert(string text, int nRows)來將傳入的字串text轉換成n行的Z字轉換。

convert("PAYPALISHIRING", 3) 會回傳 "PAHNAPLSIIGYIR"。

這邊用另外一個範例來解釋會比較清楚：
text = "ABCDEFGHIJKLMN", n = 4，排成Z字如下，因此轉換後的字串爲 "AGMBFHLNCEIKDJ"

```
A     G     M
B   F H   L N
C E   I K
D     J
```

思路

1. 從上面nRows=3，nRows=4觀察可以知道，字母要回到第一列總共要經過 2*nRows - 2*次變化，例如上面的例子*A->G*之間總共經過 *2*4-2 = 6個字母
2. 用一個array來儲存每一列的字母，上面範例最後array[0] = 'AGM'
3. n = 2*nRows - 2代表實際上經過的字母數，字母位置爲i， i%n < nRows時，直接將字母放入array[i]，

例如"B"的位置i爲1，1%6 = 1， 將"B"放入array[1]

4. 當i%n >= nRows時， 放入的位置爲 nRows-1 - (i%n) - nRows -1，整理一下可以寫成 2*nRows - i%n - 2*
   例如上面的*"F" [i=5, i%6 = 5], 5 > 4* 因此*"F"*在*array*中的位置爲 *2*4 - 5 - 2 = 1，因此將"F"放到array[1]，也就是array[1] = "BF"
5. 最後將array內的字串結合回傳

解題

```
/**
 * @param {string} s
 * @param {number} numRows
 * @return {string}
 */
var convert = function(s, numRows) {
    if(s == null) return "";
    if(numRows == 1)  return s;

    // 每一輪的變化總共是numRows*2 - 2種
    var n = numRows*2 - 2;
    var array = [];

    // 創立一個有numRows元素的array
    for(var k = 0 ; k < numRows; k++){
        array.push("");
    }

    for(var i in s){
        var lineNumber = i%n;
        if(lineNumber < numRows){
            // i%n 比 numRows小，s[i]直接放入array
            array[lineNumber] += s[i];
        } else {
            // 計算s[i]應該屬於array哪個位子
            //array[numRows-1 - lineNumber-numRows-1] += s[i];
            array[2*numRows - lineNumber -2] += s[i];
        }
    }

    return array.join("");
};
```

LeetCode 125. Valid Palindrome

LeetCode 125. Valid Palindrome

題目

Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

For example, "A man, a plan, a canal: Panama" is a palindrome. "race a car" is not a palindrome.

Note: Have you consider that the string might be empty? This is a good question to ask during an interview.

For the purpose of this problem, we define empty string as valid palindrome.

翻譯

給一個字串，不考慮大小寫與非字母數字的情況下，判斷這個字串是不是迴文。

範例：
"A man, a plan, a canal: Panama" --> true
"race a car" --> false (raceacar != racaecar)

注意：
你有考慮過空字串的情況嗎，在這邊我們定義空字串是一個迴文。

思路

這題我覺得頗簡單，不知道爲啥通過率這麼低。

1. 首先把字串轉爲全小寫
2. 將字串裡面不是字母與數字的部分去除
3. 反轉字串判斷與上一個步驟處理過的字串相等

解題

```
/**
 * @param {string} s
 * @return {boolean}
 */
var isPalindrome = function(s) {
    //轉小寫
```

```
    s = s.toLowerCase();
    //取代非文字部分
    s = s.replace(/[^a-z0-9]/ig,"");
    //反轉
    var rev = s.split("").reverse().join("");
    //判斷反轉後是否與之前處理過的字串相等
    return s.indexOf(rev) == 0;
};
```

LeetCode 165. Compare Version Numbers

LeetCode 165. Compare Version Numbers

題目

Compare two version numbers version1 and version2. If version1 > version2 return 1, if version1 < version2 return -1, otherwise return 0.

You may assume that the version strings are non-empty and contain only digits and the . character. The . character does not represent a decimal point and is used to separate number sequences. For instance, 2.5 is not "two and a half" or "half way to version three", it is the fifth second-level revision of the second first-level revision.

Here is an example of version numbers ordering:

```
0.1 < 1.1 < 1.2 < 13.37
```

翻譯

比較兩個版本號，version1與version2，如果version1 > version2 回傳1，如果version2 > version1 回傳-1，相等的話回傳0。

你可以假設版本號裡面只有數字跟"."，"."在這邊不是表示小數點而是用來分割子版本的序號。
舉例來說，2.5不是數字2+0.5，而是第2.5版。

下面是一些範例

```
0.1 < 1.1 < 1.2 < 13.37
```

思路

版本號是用數字與"."組成，因此先將版號用"."分開放在陣列中，接著就可以跑迴圈比對，如果相同位置下， version1數字比version2大就表示 version1 > version2，如果到迴圈結束都比不出大小，兩個版號相等。

其實這題也可以用純字串來解，速度會快很多，不過因爲寫這題的時候我才剛接觸leetcode沒多久，覺得用陣列解會比較清楚。

解題

```
/**
```

```
 * @param {string} version1
 * @param {string} version2
 * @return {number}
 */
var compareVersion = function(version1, version2) {
            // 將版本號用小數點切開
            var array1 = version1.split(".");
            var array2 = version2.split(".");

            // 取版本號較長的跑loop
            var max = array1.length > array2.length ? array1.length:a

            for(var i = 0 ; i < max ; i++ ){
                var i1 = parseInt(array1[i]);
                var i2 = parseInt(array2[i]);

                // 如果其中一邊後面已經沒版本號，後面就設爲0
                // ex.  [1.0.1], [1.0] 轉換成 [1.0.1], [1.0.0]作比對
                i1 = array1.length < i+1 ? 0 : i1;
                i2 = array2.length < i+1 ? 0 : i2;


                // 如果相同位置的版號數字不一樣，就可以判斷哪個版號大
                // ex. 1.1.5 < 1.2.2，因爲在第二個位置時 2>1 就可以得到答
                if(i1 > i2){
                    return 1;
                }
                if(i2 > i1){
                    return -1;
                }
            }
            return 0;
};
```

LeetCode 8. String to Integer (atoi)

LeetCode 8. String to Integer (atoi)

題目

Implement atoi to convert a string to an integer.

Hint: Carefully consider all possible input cases. If you want a challenge, please do not see below and ask yourself what are the possible input cases.

Notes: It is intended for this problem to be specified vaguely (ie, no given input specs). You are responsible to gather all the input requirements up front.

Requirements for atoi:
The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in str is not a valid integral number, or if no such sequence exists because either str is empty or it contains only whitespace characters, no conversion is performed.

If no valid conversion could be performed, a zero value is returned. If the correct value is out of the range of representable values, INT_MAX (2147483647) or INT_MIN (-2147483648) is returned.

翻譯

實作atoi將字串轉成int。

提示：小心仔細的考慮所有可能的輸入，如果你想挑戰自己，不要看下面的文字，直接想可能的輸入有哪些就可以開始寫了。

注意：這題目的輸入值有各種組合，你要先收集任何可能的輸入。

atoi的需求：
首先輸入的開頭可能是一連串的空白，因此要先找到第一個非空白字元。然後從這個字元開始可能會有正負號在數字的前面，將這些字串轉換成數字。

如果在數字後面有出現其他非數字的符號，因爲他們對值沒有影響，可以忽略這些符號。

如果第一個非空白字元不是一個合法的int整數，或者字串裡面都是空白字元，那也視爲不合法的輸入。

不合法的輸入回傳0，如果轉換後的數字num > INT_MAX (2147483647) 回傳2147483647，num < INT_MIN (-2147483648) 回傳 -2147483648

思路

這題看起來很複雜，通過率也低的很可怕，莫驚莫怕莫慌張，其實慢慢的一邊改一邊做，經過大量的Trial & Error，爲拉低通過率做出一點貢獻，最後還是解得出來。

這邊當然不是來說一句Trial & Error，這題可以拆成以下幾個步驟一步一步解開：

1. 先把字串前面可能出現的空白字串都消除，找到第一個出現得+-符號或數字。
2. 如果剩下的開頭是+-符號，先用一個變數存放起來，然後把他截掉。
3. 如果輸入的字串是合法的，理論上現在開頭就應該只剩下數字，如果又出現+-號或其他字元，就可以判定爲不合法字串。
4. 經過上面3取到最後都是數字，不過這沒影響。關，剩下的開頭肯定是數字了，這代表我們一定可以解析出一個整數出來，接下來將開頭到不是數字字元中間的數字字串取出來，當然也有可能一直到最後都是數字，不過這沒影響。
5. 結合剛才儲存的正負符號與上一個步驟拿到的數字字串後轉換成整數，這時候還要判斷整數是否超過int的最大值(2147483647)與最小值(-2147483648)，超過的話上面題目已經說明過要怎麼處理，沒超過就直接回傳解析後的整數。

解題

```
/**
 * @param {string} str
 * @return {number}
 */
var myAtoi = function(str) {
    var sign = '+'; //正負號
    var numReg = /^[0-9]/;

    //將前面的空白字串去掉,  ex. "^^^^-122a" -> "-122a" (^代表空白)
    var i = 0;
    while(str.charAt(i)==' ' && i < str.length){
        i++;
    }
    str = str.slice(i);


    //處理正負號, ex. "-122a" -> "122a"
    if(str.startsWith('+')){
        str = str.slice(1);
    } else if(str.startsWith('-')){
        str = str.slice(1);
        sign = '-';
    }
```

```
    //處理後的字串不是數字開頭,代表字串不合法
    if(!numReg.test(str)) return 0;

    //出現不是數字的符號就中斷
    var j = 0;
    while(j < str.length && numReg.test(str[j])){
        j++;
    };
    // 截出開頭的數字字串, ex. '122a' -> '122'
    str = str.substr(0,j);

    //字串轉成int
    var value = parseInt(sign+str);
    if(value > 2147483647){
        return 2147483647;
    }
    if(value < -2147483648){
        return -2147483648;
    }
    return value;
};
```

LeetCode 258. Add Digits

LeetCode 258. Add Digits

題目

Given a non-negative integer num, repeatedly add all its digits until the result has only one digit.

For example:

Given num = 38, the process is like: 3 + 8 = 11, 1 + 1 = 2. Since 2 has only one digit, return it.

Follow up:
Could you do it without any loop/recursion in O(1) runtime?

翻譯

將一個數字每個位數相加，直到剩個位數爲止。

範例：
num = 38，則 3+8 = 11，1+1 = 2, 2是個爲數，回傳2。

進階：
不用迴圈，遞迴解這個問題

思路

取出每個位數的值，例如說38 -> [3,8]，然後將取出的數字相加，如果大於等於10，重複動作直到數字小於10。

解題

方法一

```
var addDigits = function(num) {
    while(num >= 10){
       var n = num;
       var sum = 0;
       // 取出每個位數的值
       // ex. n = 138, 138%10 = 8, sum = 8, n = 13
       //     n = 13,  13%10  = 3, sum =11, n = 1
       while(parseInt(n/10) > 0){
```

```
            sum += parseInt(n%10);
            n = parseInt(n/10);
        }
        // sum = 11, n = 1, sum + n%10 = 12,
        num = parseInt(n%10) + sum;
    }
    return num;
};
```

方法二

```
var addDigits = function(num) {
    while(num >= 10){
        var sum = 0;
        // 將數字轉成字串並切開放在陣列 ex. 138 => [1,3,8]
        (''+num).split('').forEach(function(v){
            sum += parseInt(v);
        });
        num = sum;
    return num;
};
```

進階

這已經是偏向數學的範圍，怎麼證明這邊就跳過了，麻煩自己上網搜尋， 簡單說就是判斷一個數是否爲9的倍數，可以從每個位數相加是否能被9整除直接判斷，運用這樣的想法，直接取這個數除9的餘數。

```
var addDigits = function(num) {
    if(num == 0) return 0;
    if(num%9==0) return 9;
    return num%9;
};
```

LeetCode 171. Excel Sheet Column Number

LeetCode 171. Excel Sheet Column Number

題目

Related to question Excel Sheet Column Title

Given a column title as appear in an Excel sheet, return its corresponding column number.

For example:

```
A -> 1
B -> 2
C -> 3
...
Z -> 26
AA -> 27
AB -> 28
```

翻譯

將Excel欄位轉換成數字。

思路

1. A-Z總共26個字母，因此這就是一個26進位的系統
2. 將字串分別取出字元A-Z，根據ANSI CODE，A的code爲65，A = 65 - 64 = 1
3. 以AB爲例， AB = (A)*26^1 + (B)26^0 = 1*26+ 2*1 = 28
4. 以AZ爲例， AZ = (A)*26^1 + (Z)26^0 = 1*26+ 26*1 = 52

解題

```
/**
 * @param {string} s
 * @return {number}
 */
var titleToNumber = function(s) {
    var sum = 0;
    var exp = 0;
    for(var i = s.length -1 ; i >= 0 ; i--){
        // 根據ansi將字元轉成數字，這邊是從字串後面，也就是低位數開始取
```

```
        var v = s.charCodeAt(i) - 64;
        // 每多一個字元代表26的n次方
        v = v*Math.pow(26,exp++);
        sum += v;
    }
    return sum;
};
```

LeetCode 326. Power of Three

LeetCode 326. Power of Three

題目

Given an integer, write a function to determine if it is a power of three.

Follow up: Could you do it without using any loop / recursion?

翻譯

判斷一個整數是否是3的次方數。

進階： 不使用迴圈，遞迴解題?

思路

不管進階的話跟LeetCode 231. Power of Two解法是一模一樣的。 因此只要不斷的把這個數除3，如果發現餘數不是0，就可以判斷這個數不是3的次方數

解題

```
/**
 * @param {number} n
 * @return {boolean}
 */
var isPowerOfThree = function(n) {
    while(n>2){
        if(n%3 !== 0) return false;
        n = parseInt(n/3);
    }

    return n==1;
};
```

LeetCode 231. Power of Two

LeetCode 231. Power of Two

題目

Given an integer, write a function to determine if it is a power of two.

翻譯

判斷一個整數是否是2的次方數。

思路

只要n大於3不斷的將輸入值n除2，如果發現餘數不等於0，那他就不是2的次方數。

解題

```
/**
 * @param {number} n
 * @return {boolean}
 */
var isPowerOfTwo = function(n) {
    if(n <= 0) return false;
    if(n === 1) return true;
    while(n>3){
        if(n%2 != 0) return false;
        n = parseInt(n/2);
    }
    return n%2 === 0;
};
```

Plus

例用javascript的特性，稍爲改寫一下，增加效能

```
/**
 * @param {number} n
 * @return {boolean}
 */
```

```
var isPowerOfTwo = function(n) {
    if(n <= 0) return false;
    if(n === 1) return true;
    while(n>5){
        n = n/2;
    }
    return n%2 === 0;
};
```

LeetCode 202. Happy Number

LeetCode 202. Happy Number

Write an algorithm to determine if a number is "happy".

A happy number is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1. Those numbers for which this process ends in 1 are happy numbers.

Example: 19 is a happy number
1^2 + 9^2 = 82
8^2 + 2^2 = 68
6^2 + 8^2 = 100
1^2 + 0^2 + 0^2 = 1

翻譯

判斷一個數字是否爲happy number。

happy number 定義如下：當一個數的每位數平方後相加，大於1則重複每位數開平方相加的動作，如果最後得到1的話，這個數就是happy number，如果進入無窮迴圈，這個數就不是happy number。

思路

使用一個map儲存計算過的數字，如果目前的數字已經計算過，表示無窮迴圈出現，return false。持續計算到1出現true或是無窮迴圈出現false。

範例：判斷4是不是happy number

```
4^2       = 16
1^2 + 6^2 = 37
3^2 + 7^2 = 58
.....
......... = 20
2^2 + 0   = 4
```

4重複出現，因此進入無窮迴圈

解題

```
/**
```

```
 * @param {number} n
 * @return {boolean}
 */
var isHappy = function(n) {
    // map儲存計算過的數字
    var store = {};

    // 如果map裡面出現過這個數字 或 數字 = 1，停止迴圈。
    while(!store[n] && n!=1){
        store[n] = n;
        // 單純的計算每一個位數的平方和
        n.toString().split("").forEach(function(v,i){
            if(i == 0) n = 0;
            n += v*v;
        })
        n = parseInt(n);
    }
    return n == 1;
};
```

Plus

- 以下這種寫法也可以，1~9之中，只有1,7是happy number
- 如果n超過10，計算到最後一定會落在1~9之中
- 判斷計算到最後的數字是不是1

```
/**
 * @param {number} n
 * @return {boolean}
 */
var isHappy = function(n) {
    while(n>6){
        n.toString().split("").forEach(function(v,i){
            if(i == 0) n = 0;
            n += v*v;
        })
    }
    return n == 1;
};
```

LeetCode 263. Ugly Number

LeetCode 263. Ugly Number

題目

Write a program to check whether a given number is an ugly number.

Ugly numbers are positive numbers whose prime factors only include 2, 3, 5. For example, 6, 8 are ugly while 14 is not ugly since it includes another prime factor 7.

Note that 1 is typically treated as an ugly number.

翻譯

判斷一個數字是否爲ugly number

ugly number是說一個數字因式分解後只含有2,3,5這些因子，例如說6[2,3],8[2,2,2]都是ugly的，14[2,7]因爲含有7，所以不是ugly。

注意: 1 是一個ugly number (LeetCode 202. Happy Number這題可以看到，1也是一個happy number，可知道1是一個醜但是樂觀的數字 XD)

思路

一開始我是想用先排出一個在n之下的質數表，用這個表來因數分解，不過這種寫法會超出時間。

重新想了一下，發現單純用除法可以解，一個n如果小於7，一定是ugly number，如果大於等於7，將n裡面的2,3,5因子除掉，最後如果還有剩下其他因子，這個數字就不是ugly number。

解題

```
/**
 * @param {number} num
 * @return {boolean}
 */
var isUgly = function(num) {
    if(num <= 0){
        return false;
    }
    if(num <= 6) return true;
```

```
    while(num > 2){
        if(num%2 !== 0){
            break;
        }
        num = parseInt(num/2);
    }

    while(num > 3){
        if(num%3 !== 0){
            break;
        }
        num = parseInt(num/3);
    }

    while(num > 5){
        if(num%5 !== 0){
            break;
        }
        num = parseInt(num/5);
    }
    return num%2===0 || num % 3 === 0 || num%5 ===0;
}
```

LeetCode 342. Power of Four

LeetCode 342. Power of Four

題目

Given an integer (signed 32 bits), write a function to check whether it is a power of 4.

Example: Given num = 16, return true. Given num = 5, return false.

Follow up: Could you solve it without loops/recursion?

翻譯

判斷一個32bits的int整數是否爲4的倍數。

進階：不用迴圈，遞迴解題。

思路

一直除4，如果出現餘數，就不是4的整數。

解題

```
/**
 * @param {number} num
 * @return {boolean}
 */
var isPowerOfFour = function(num) {
    if(num <= 0 ) return false;
    if(num   == 1) return true;
    if(num%2 == 1) return false;

    while(num >= 4){
        if(num%4!=0) return false;
        num = parseInt(num/4);
    }

    return num==1 ;
};
```

LeetCode 66. Plus One

LeetCode 66. Plus One

題目

Given a non-negative number represented as an array of digits, plus one to the number.

The digits are stored such that the most significant digit is at the head of the list.

翻譯

給一包含非數整數的陣列，其中每一個值代表該位數的值，對這個陣列加1。

範例：
19 = [1,9] ， 19+1 = 20 = [2,0]。

思路

1. 陣列最後一個數是個位數，所以從後面開始讀，個位數+1後，如果有進位，儲存進位值，val = 0，沒進位直接存儲。
2. 處理十位數，如果個位數有進位，十位數+1，再判斷十位數是否有進位
3. 重複上面動做值到陣列結束

解題

```
/**
 * @param {number[]} digits
 * @return {number[]}
 */
var plusOne = function(digits) {
    // 判斷相加後是否需進位
    var carry = 0;

    for(var i = digits.length - 1 ; i >= 0 ; i--){
        // 目前位數 = 目前位數+前面是否進位
        digits[i]= digits[i] + carry;

        // list最後一個數字，也就是個位數，給他+1
        if(i == digits.length - 1 ){
          digits[i] = digits[i] + 1;
        }
```

```
        // 如果目前這個位數等於10(因爲只+1，所以不會超過10)，進位
        if(digits[i] == 10){
          digits[i] = 0;
          carry = 1;
        } else {
          carry = 0;
        }
    }

    // 最後如果有進位
    if(carry == 1){
        digits.unshift(carry);
    }

    return digits;
};
```

LeetCode 172. Factorial Trailing Zeroes

LeetCodee 172. Factorial Trailing Zeroes

題目

Given an integer n, return the number of trailing zeroes in n!.

Note: Your solution should be in logarithmic time complexity.

翻譯

給一個正整數n，回傳n!中有幾個0

注意：你的解法應該是log(n)的時間複雜度。

範例： n = 5 ; n! = 120 回傳 1。

思路

1. 當出現0，也就是10的n次方，可以推論一定要出現因子裡面含有2跟5的數字
2. 2這個數字到處撿都是，眞正決定會出現幾個0的，是n!裡面包含幾個5
3. 例如上面的n=5，54321 = 120，可以發現5*2 =10，因此會出現一個0
4. n = 25，會出現 25,20,15,10,5共5個帶有5的數字，不過25其實包含了5*5，所以25!總共會出現5+1=6個10。

解題

```
/**
 * @param {number} n
 * @return {number}
 */
var trailingZeroes = function(n) {
    if(n < 5) return 0 ;

    var count = 0;
    // 算階層內有幾個5出現
    while(n >= 5){
        count += Math.floor(n/5);
        n = parseInt(n/5);
    }
```

```
    return count;
};
```

LeetCode 9. Palindrome Number

LeetCodee 9. Palindrome Number

題目

Determine whether an integer is a palindrome. Do this without extra space.

Some hints: Could negative integers be palindromes? (ie, -1)

If you are thinking of converting the integer to string, note the restriction of using extra space.

You could also try reversing an integer. However, if you have solved the problem "Reverse Integer", you know that the reversed integer might overflow. How would you handle such case?

There is a more generic way of solving this problem.

翻譯

判斷一個int整數是否是自己的迴文數，不能使用額外的空間來操作。

提示：
負整數會是自己的迴文數嗎(ex. -1)

如果你想用字串來解是不行的，因爲不能使用額外的空間。

你也可以反轉整數，如果你之前已經做過LeetCode 7. Reverse Integer，你會知道反轉後的數可能會超過integer的最大值。

思路

不使用額外空間的意思，根據discuss裡面的討論，應該是不能用一個O(n)的額外空間(ex. array, string之類的)，用一個O(1)的變數是可以的。 把傳入的x整個反轉後跟本來的x比較是否一致，這題還算簡單，直接看code。

解題

```
/**
 * @param {number} x
 * @return {boolean}
 */
var isPalindrome = function(x) {
    // x < 0 or x > (2^32 - 1) , false;
```

```
    if(x < 0 || x > Math.pow(2,32)-1) return false;
    if(x < 10) return true;

    // keep x
    var num = x;

    // 先抓出最高位數，轉成個位數
    var recNum = x%10;
    x  = parseInt(x/10);

    // 將recNum*10，再抓最高位數加到recNumb
    while(x != 0){
        recNum = recNum*10;
        recNum = recNum + x%10;
        x = parseInt(x/10);
    }
    return recNum == num;
};
```

LeetCode 7. Reverse Integer

LeetCode 7. Reverse Integer

題目

Reverse digits of an integer.

Example1: x = 123, return 321 Example2: x = -123, return -321

Have you thought about this? Here are some good questions to ask before coding. Bonus points for you if you have already thought through this!

If the integer's last digit is 0, what should the output be? ie, cases such as 10, 100.

Did you notice that the reversed integer might overflow? Assume the input is a 32-bit integer, then the reverse of 1000000003 overflows. How should you handle such cases?

For the purpose of this problem, assume that your function returns 0 when the reversed integer overflows.

翻譯

反轉一個int整數。

x = 123 , return 321 x = -123 , return -321

提示：
假如10，100反轉後會長怎樣。

你有注意到反轉後的數可能會超過Integer的範圍嗎，例如說1000000003反轉後就超過了32-bit的integer。這種情況要怎麼處理?

在這個問題中，超過integer只要回傳0就可以。

思路

這是我一開始寫leetCode的解法，把數字轉成string後反轉，要額外處理的就是開頭是負數，-123反轉後變成321-，需把最後的負號搬到前面。

另外有不使用額外空間(ex. array,string etc.)的解法，請參考 9. Palindrome Number

解題

```
/**
 * @param {number} x
 * @return {number}
 */
var reverse = function(x) {
    var INT_MAX = Math.pow(2,31)-1;
    if(0 <= x && x < 10) return x;

    var nFlag = "";
    // x to string
    var str = x.toString();

    // reverse number string
    var rStr = nFlag + str.split("").reverse().join("");

    // if x < 0, move '-'  from rStr back to front
    if(rStr.indexOf('-') != -1){
        rStr = '-' + rStr.replace('-','');
    }

    var result = parseInt(rStr);

    if(result > INT_MAX || result < -(1+INT_MAX)) return 0;
    return result;
};
```

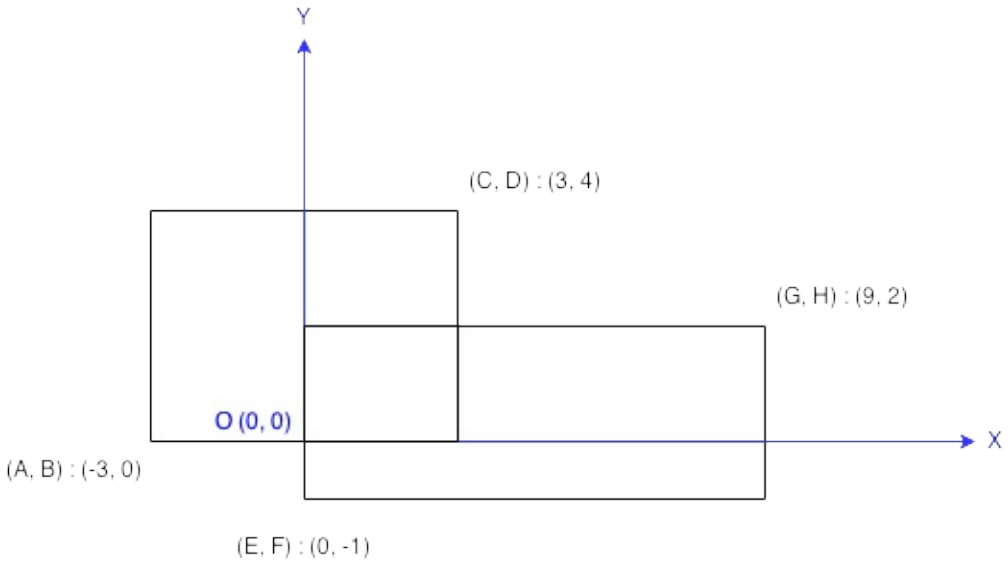
LeetCode 223. Rectangle Area

223. Rectangle Area

題目

Find the total area covered by two rectilinear rectangles in a 2D plane.

Each rectangle is defined by its bottom left corner and top right corner as shown in the figure.



Assume that the total area is never beyond the maximum possible value of int.

翻譯

計算兩個長方形所覆蓋的面積，每個長方形都是由左下頂點與右上頂點決定，如圖所示。

假設覆蓋面積不會過int的最大值。

思路

1. 先考慮兩個長方形不交疊的情況，只要單純的計算面積相加即可
2. AB,EF分別爲左下頂點，CD,GH分別爲右上頂點，如果A>=G表示第一個長方形在第二個長方形右側而且面積不重疊
3. 同樣方法可判斷兩個長方形其他三個點是否有交疊的情況
4. 如果有交疊的情況發生，使用max(A,E)可以找出交疊正方形的左下頂點，同樣方法可以找出交疊正方形正確位置並計算面積

解題

```
/**
 * @param {number} A
 * @param {number} B
 * @param {number} C
 * @param {number} D
 * @param {number} E
 * @param {number} F
 * @param {number} G
 * @param {number} H
 * @return {number}
 */
var computeArea = function(A, B, C, D, E, F, G, H) {
    // 分別計算ABCD與EGFH的面積
    var r1 =  Math.abs(A-C)*Math.abs(B-D);
    var r2 =  Math.abs(E-G)*Math.abs(F-H);

    // 如果ABCD與EGFH沒重疊，直接將r1,r2加總
    if( A >= G || B >= H || C <= E || D <= F){
        return r1+r2;
    }

    // 計算重疊的部分
    var rD = Math.abs( (Math.max(A,E) - Math.min(C,G)) * (Math.max(B,

    return r1+r2 - rD;
};
```

LeetCode 204. Count Primes

LeetCode 204. Count Primes

題目

Description:

Count the number of prime numbers less than a non-negative number, n.

翻譯

給一個n，計算比n小的質數有幾個。

思路

1. 網路上解質數問題的演算法很多，這邊用比較原始的方法
2. 判斷n之下有幾個質數，只要跑迴圈判斷從2~(n-1)中每一個數是不是質數就可以
3. 質數p的定義就是 p/2 , p/3 , p/4 .... p/(p-1)都不等於0
4. 實作上不需要除到p-1，只需要除到"p的平方根"就可以，而且可以跳過2的倍數 p/2, p/3 , p/5 , p/7 .....

解題

```
/**
 * @param {number} n
 * @return {number}
 */
var countPrimes = function(n) {
    // n = 3的時候，才會出現第一個比n小的質數2
    if(n < 3) return 0;

    var count = 1;
    // 加快速度，所以跳過2的倍數
    for(var i = 3 ; i < n ; i+=2){
        var flag = true;
        // 判斷i是不是質數
        for(var j = 3 ; j*j <= i; j+=2){
            if(i%j == 0){
                // i能被比自己小的數除盡，表示i不是質數
                flag = false;
                break;
```

```
            }
        }

        if(flag) count++;
    }

    return count;
};
```

LeetCode 168. Excel Sheet Column Title

LeetCode 168. Excel Sheet Column Title

題目

Given a positive integer, return its corresponding column title as appear in an Excel sheet.

For example:

```
    1 -> A
    2 -> B
    3 -> C
    ...
    26 -> Z
    27 -> AA
    28 -> AB
```

翻譯

給一個正整數，回傳在excel表格中對應的欄位。

思路

下表的來源

```
A   1     AA    26+ 1     BA  2×26+ 1     ...     ZA  26×26+ 1     A
B   2     AB    26+ 2     BB  2×26+ 2     ...     ZB  26×26+ 2     A
.   .     ..    .....     ..  .......     ...     ..  ........     .
.   .     ..    .....     ..  .......     ...     ..  ........     .
.   .     ..    .....     ..  .......     ...     ..  ........     .
Z  26     AZ    26+26     BZ  2×26+26     ...     ZZ  26×26+26     A
```

觀察上面表格可以知道數字轉換成excel欄位是一個類似27進位的轉換，不過規則有一點點不同。

ABC = 1x26^2 + 2x26^1 + 3 ，看起來可以用 String.fromCharCode(n%26 + 64)直接將數字轉成英文字母
ZZZ =26x26^2 + 26x26^1 + 26，可是當n=26，也就是字母Z的時候，26%26 = 0，反而不能得到正確的字母Z
因此改用***String.fromCharCode((n-1)%26 + 65)***處理

解題

```
/**
 * @param {number} n
 * @return {string}
 */
var convertToTitle = function(n) {
    // String.fromCharCode(65) = 'A'， String.fromCharCode(66) = 'B'

    // 比27小的時候，可以直接用String.fromCharCode(n)轉換成英文字母
    if(n-1 < 26){
        return String.fromCharCode(65+(n-1)%26);
    }

    var result = "";

    while(n > 0){
        var codeNum = (n-1)%26;
        var c = String.fromCharCode(codeNum+65);
        result = c + result;

        n = parseInt((n-1) / 26);
    }
    return  result;
};
```

亂湊出來的解法

```
/**
 * @param {number} n
 * @return {string}
 */
var convertToTitle = function(n) {
    // String.fromCharCode(65) = 'A'， String.fromCharCode(66) = 'B'
    // 比27小的時候，可以直接用String.fromCharCode(n)轉換成英文字母
    if(n < 27){
        return String.fromCharCode(64+n%27);
    }

    var result = "";

    while(n > 0){
        var c = ""

        if(n%26 == 0) {
            // n=26時候代表的字母應該爲'Z'，不過n%26 = 0，因此直接用n%26+64
            c = String.fromCharCode(90);
            n  = parseInt(n/26)-1;
        } else {
            // 'A' = charCode(1+64)，剩下的字母一樣推算
            c = String.fromCharCode(n%26+64);
            n  = parseInt(n/26);
```

```
        }
        result = c + result;
    }
    return  result;
};
```

LeetCode 283. Move Zeroes

LeetCode 283. Move Zeroes

題目

Given an array nums, write a function to move all 0's to the end of it while maintaining the relative order of the non-zero elements.

For example, given nums = [0, 1, 0, 3, 12], after calling your function, nums should be [1, 3, 12, 0, 0].

Note:
You must do this in-place without making a copy of the array.
Minimize the total number of operations.

翻譯

給一個陣列，把裡面出現的0搬到陣列最後面，剩下的元素保持原本的排序。

範例：
nums = [0, 1, 0, 3, 12] ，執行後回傳[1, 3, 12, 0, 0]

注意：
只能在當前陣列中操作，不能用一個新的陣列來解。盡量減少操作次數。

思路

這題用到的技巧後面有不少題目一樣會用到，使用另外一個變數index來記錄碰到的非0數字，遇到非0整數，將整數塞到index的位子， 最後陣列長度扣除index就是0的數目，將陣列index之後的數字改爲0就完成搬移的動作。

ex. nums = [0, 1, 0 , 3] , index = 0
nums[0] == 0, 跳過
nums[1] == 1, nums[0] = 1, index++;
nums[2] == 0, 跳過
nums[3] == 3, nums[1] = 3, index++;
結束後 nums = [1,3,0,3]，接下來將nums後面的[0,3]用0取代

解題

```
/**
 * @param {number[]} nums
 * @return {void} Do not return anything, modify nums in-place instea
```

```
 */
var moveZeroes = function(nums) {
    var index = 0;
    for(var i = 0 ; i < nums.length ; i++){
        var n = nums[i];
        // not zero, index++, push to array
        if(n !== 0){
            nums[index++] = n;
        }
    }

    // after index to zero
    for(index ; index < nums.length ; index++){
        nums[index] = 0;
    }
};
```

LeetCode 349. Intersection of Two Arrays

LeetCode 349. Intersection of Two Arrays

題目

Given two arrays, write a function to compute their intersection.

Example:
Given nums1 = [1, 2, 2, 1], nums2 = [2, 2], return [2].

Note:
Each element in the result must be unique.
The result can be in any order.

翻譯

尋找兩個陣列的交集。

範例：
nums1 = [1, 2, 2, 1]，nums2 = [2, 2]，return [2]。

注意：
同樣的數字只出現一次，回傳的陣列內容不限排序。

思路

1. 先判斷nums1與nums2長度誰比較短，我們就稱它叫ary拿來跑迴圈，較長的陣列叫store。
2. ary[i]如果可以在store中找到值，表示這是交集的數字。
3. 判斷結果陣列(result)是否已經有交集數字，如果沒有就新增到結果陣列(result)中。

解題

```
/**
 * @param {number[]} nums1
 * @param {number[]} nums2
 * @return {number[]}
 */
var intersection = function(nums1, nums2) {
    var result = [];
    var store;  // 長array
```

```
    var ary;    // 短array

    // 判斷nums1,nums2長度
    if(nums1.length > nums2.length){
        store = nums1;
        ary = nums2;
    } else {
        store = nums2;
        ary = nums1;
    }

    // 只需需要跑較短的array就行
    for(var i = 0 ; i < ary.length ; i++){
        var value = ary[i];
        // 如果可以在長array中找到目前的值，而且在結果array中找不到，代表這個
        if(store.indexOf(value) >= 0 && result.indexOf(value) == -1)
            result.push(value);
        }
    }
    return result;
};
```

LeetCode 169. Majority Element

LeetCode 169. Majority Element

題目

Given an array of size n, find the majority element. The majority element is the element that appears more than ⌊ n/2 ⌋ times.

You may assume that the array is non-empty and the majority element always exist in the array.

翻譯

長度爲n的陣列，找出一個出現n/2次以上的主要元素，假設陣列不會是空值，而且總是會有主要元素存在陣列中。

思路

用一個map來記錄每一個元素出現幾次，只要有一個元素出現n/2次，這個元素就是主要元素。

解題

```
/**
 * @param {number[]} nums
 * @return {number}
 */
var majorityElement = function(nums) {
    var map = {};
    if(nums.length ===1) {return nums[0]}

    for(var i = 0 ; i < nums.length ; i++){
        // 第一次出現的元素，放到map    not in map, push element and set
        if(!map[nums[i]]) {
            map[nums[i]] = 1;
        } else {
            // 出現過的元素，數量增加    element exist, count++
            map[nums[i]]++;
            // 判斷這個元素出現的次數 > n/2
            if(map[nums[i]] >= nums.length/2){
                return nums[i];
            }
```

```
            }
        }
};
```

LeetCode 217. Contains Duplicate

LeetCode 217. Contains Duplicate

題目

Given an array of integers, find if the array contains any duplicates. Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

翻譯

給一個陣列，確認陣列中是否有重複的元素

思路

解題方向與169. Majority Element完全相通，使用map，當一個元素count = 2時，回傳true，迴圈如果跑完回傳false。

解題

```
/**
 * @param {number[]} nums
 * @return {boolean}
 */
var containsDuplicate = function(nums) {
    var keep = [];
    for(var i in nums){
        if(keep.indexOf(nums[i])<0){
            keep.push(nums[i]);
        } else {
            return true;
        }
    }
    return false;
};
```

LeetCode 350. Intersection of Two Arrays II

LeetCode 350. Intersection of Two Arrays II

題目

Given two arrays, write a function to compute their intersection.

Example:
Given nums1 = [1, 2, 2, 1], nums2 = [2, 2], return [2, 2].

Note:

- Each element in the result should appear as many times as it shows in both arrays.
- The result can be in any order.

Follow up:

- What if the given array is already sorted? How would you optimize your algorithm?
- What if nums1's size is small compared to nums2's size? Which algorithm is better?
- What if elements of nums2 are stored on disk, and the memory is limited such that you cannot load all elements into the memory at once?

翻譯

尋找兩個陣列的交集。

範例：
nums1 = [1, 2, 2, 1]，nums2 = [2, 2]， return [2,2]

注意：

- 同樣的數字在回傳的陣列中可重複出現
- 回傳的陣列可以不管裡面的數字排序

進階：

- 如果陣列已經排序過，如何最佳化演算法?
- 如果nums1長度小於nums2長度，如何最佳化演算法?
- 如果nums2儲存在光碟上，所以無法一次讀盡所有的元素，要怎麼處理?

思路

不管進階的部分，這題跟LeetCode 349. Intersection of Two Arrays很像，只是交集的數字可重複

出現。

先找出哪個陣列較長爲store，較短的爲ary，ary[i]如果可以在store中找到值，表示這是交集的數字，將這數字放入回傳陣列，移除store中第一個交集數字。

解題

```
var store, array;
var number = [];
if(nums1.length > nums2.length){
    store = nums1;
    array = nums2;
} else {
    store = nums2;
    array = nums1;
}

for(var i = 0; i < array.length ; i++){
    var v = array[i];
    if(store.indexOf(v) >= 0){
        store[store.indexOf(v)] = null;
        number.push(v);
    }
}
return number;
```

LeetCode 27. Remove Element

LeetCode 27. Remove Element

題目

Given an array and a value, remove all instances of that value in place and return the new length.

Do not allocate extra space for another array, you must do this in place with constant memory.

The order of elements can be changed. It doesn't matter what you leave beyond the new length.

Example: Given input array nums = [3,2,2,3], val = 3

Your function should return length = 2, with the first two elements of nums being 2.

翻譯

給一個陣列跟一個數字，移除陣列中所有跟數字相同的元素。

不可以使用另外的陣列來處理，全部的操作都要在同一個陣列中。

陣列中的元素可以隨意排序。

範例：
`nums = [3, 1, 2, 3, 2]， val = 3`
應該要`return` 陣列的長度`3`，因爲裡面的`3`被移除後剩`[1,2,2].`

思路

這跟 LeetCode 283. Move Zeroes 很像，差別在於283移除的是0，這題移除的是特定數字，如果283有寫出來，這題應該難不倒你。

解題

```
/**
 * @param {number[]} nums
 * @param {number} val
 * @return {number}
 */
var removeElement = function(nums, val) {
    if(nums.length == 0) return nums.length;
    if(nums.indexOf(val) < 0) return nums.length;
```

```
    var count = 0;
    for(var i = 0, max = nums.length; i<max ; i++){
        if(nums[i] != val) {
            nums[count++] = nums[i];
        }
    }

    return count;
};
```

LeetCode 118. Pascal's Triangle

LeetCode 118. Pascal's Triangle

題目

Given numRows, generate the first numRows of Pascal's triangle.

For example, given numRows = 5, Return

```
[
     [1],
    [1,1],
   [1,2,1],
  [1,3,3,1],
 [1,4,6,4,1]
]
```

翻譯

numRows爲列數，產生一個Pascal三角形，例如說 numRows = 5 ，產生三角形如上所示。

思路

1. 每一列第一個值都是1。
2. 每一列第n個值則是上一列n-1位子+n位子的值。
3. 假如是該列最後一個值Nx，前一列沒有Nx這個值，可以視爲0。
4. 有了以上規則，要算出每一列的值就很簡單了，直接看下面程式碼。

解題

```
/**
 * @param {number} numRows
 * @return {number[][]}
 */
var generate = function(numRows) {
    if(numRows == 0) return [];

    // 放入第一列
    var trigle = [[1]];

    for(var i = 1 ; i < numRows ; i++){
```

```
        var prevRow = trigle[i-1]; // 前一列
        var curRow  = [1];         // 每一列開始都是1

        for(var j = 1 ; j <= i; j++){
            // 每一列的第n個值都是 前一列pre[n-1] + pre[n]
            var pre =  prevRow[j-1];
            var cur =  prevRow[j] ?  prevRow[j] : 0;
            curRow.push(pre+cur);
        }
        trigle.push(curRow);
    }

    return trigle;
};
```

LeetCode 26. Remove Duplicates from Sorted Array

LeetCode 26. Remove Duplicates from Sorted Array

題目

Given a sorted array, remove the duplicates in place such that each element appear only once and return the new length.

Do not allocate extra space for another array, you must do this in place with constant memory.

For example, Given input array nums = [1,1,2],

Your function should return length = 2, with the first two elements of nums being 1 and 2 respectively. It doesn't matter what you leave beyond the new length.

翻譯

給一個排序過的陣列，移除重複的值，每個元素只能留下一個。

不能使用其他的陣列空間，必需在本來的陣列中操作。

範例： [1,1,2] 去除重複的1之後，剩下[1,2]，回傳陣列的長度2。

思路

1. 這跟 LeetCode 283. Move Zeroes 很像，差別在於283移除的是0，這題移除的是重複的數字。
2. 使用一個count來紀錄有多少不重複的元素。
3. 當陣列中下一個元素與當前的元素重複，就跳過當前的元素，不重複就放在陣列中。

解題

```
/**
 * @param {number[]} nums
 * @return {number}
 */
var removeDuplicates = function(nums) {
    if(nums == null || nums.length == 0) return 0;
```

```
    if(nums.length == 1) return 1;
    var count = 0;
    for(var i = 1 ; i < nums.length ; i++){
        if(nums[count] != nums[i]){
            count++;
            nums[count] = nums[i];
        }
    }
    return ++count;
};
```

LeetCode 119. Pascal's Triangle II

LeetCode 119. Pascal's Triangle II

題目

Given an index k, return the kth row of the Pascal's triangle.

For example, given k = 3, Return [1,3,3,1].

Note: Could you optimize your algorithm to use only O(k) extra space?

翻譯

給一個指標k，回傳第k階的Pascal's三角形。

範例：
k=3，回傳[1,3,3,1]。

注意：你能改善演算法只用O(k)的額外空間嗎?

思路

不考慮只使用O(k)額外空間的情況下，只要參考LeetCode 118. Pascal's Triangle這題，就很簡單。

但上面的很像會超出時間，還是要用同一個陣列不斷改變裡面值的方式來做才行。每一階的Pascal's三角形都可以重上一階算出，

計算方法，每一階開頭一定是1，接下來位子n的數子爲上一階陣列pAry[n-1] + pAry[n]

```
* ex. k = 4
* 第一階  array =    [1]
* 第二階  array =   [1,1]
* 第三階  array =  [1,2,1]
* 第四階  array = [1,3,3,1]
```

解題

```
/**
 * @param {number} x
 * @return {boolean}
 */
```

```
var isPalindrome = function(x) {
/**
 * @param {number} rowIndex
 * @return {number[]}
 */
var getRow = function(rowIndex) {
    if(rowIndex == 0) return [1];
    if(rowIndex == 1) return [1,1];

    var ary = [1];

    for(var i = 1 ; i <= rowIndex ; i++){
        // 儲存 ary[n-1]的數字
        var prev = ary[i-1];
        for(var j = 1 ; j < i ; j++ ){
            // preVrow[n]的數字，如果preVrow沒有n，設爲0
            var cur = ary[j] ?  ary[j] : 0;

            // 目前row[n]的值爲 "上一個prevRow[n-1] + prevRow[n]"
            ary[j] = prev + cur;

            prev = cur;
        }
        ary.push(1);
    }
    return ary;
};
```

LeetCode 88. Merge Sorted Array

LeetCode 88. Merge Sorted Array

題目

Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.

Note: You may assume that nums1 has enough space (size that is greater or equal to m + n) to hold additional elements from nums2. The number of elements initialized in nums1 and nums2 are m and n respectively

翻譯

給兩個已經排序過的的整數陣列nums1與nums2，將nums2合併入nums1之中

注意：
nums1會有有足夠的空間可以塞入兩個陣列(nums1.length = m+n)，m爲nums1的元素數量，n爲nums2的元素數量

範例： nums1 = [1,1,2,4,6,null,null,null], m = 5, nums2 = [2,3,7], n = 3
合併後 nums1 = [1,1,2,2,3,4,6,7]

思路

1. 這題可以分成兩個步驟就會變得很簡單，先把nums2的值放到nums1裡面
   ex. nums1 = [1,2,6,null,null] , nums2 = [4,5] --> [1,2,6,4,5]
2. 然後將nums1裡面的元素重新排序 [1,2,4,5,6]

解題

```
/**
 * @param {number[]} nums1
 * @param {number} m
 * @param {number[]} nums2
 * @param {number} n
 * @return {void} Do not return anything, modify nums1 in-place inst
 */
var merge = function(nums1, m, nums2, n) {

    var index = 0;
    //將nums2裡面的值放在nums1
```

```
    for(var i = m ; i < m+n ; i++){
            nums1[i] = nums2[index];
            index++;
    }

    //使用泡沫排序法重新排序
    for(var j = 0 ; j < nums1.length - 1 ; j++){
        for(var k = j + 1 ; k < nums1.length ; k++){
            if(nums1[j] > nums1[k]){
                var temp = nums1[j];
                nums1[j] = nums1[k];
                nums1[k] = temp;
            }
        }
    }
};
```

LeetCode 219. Contains Duplicate II

LeetCode 219. Contains Duplicate II

題目

Given an array of integers and an integer k, find out whether there are two distinct indices i and j in the array such that nums[i] = nums[j] and the difference between i and j is at most k.

翻譯

給一個陣列nums跟一個整數k，判斷能不能在陣列中找到nums[i] = numsj，而且i跟j的距離不能比k還大。

範例:
nums = [1,2,3,4,1] k=3; nums[0] = nums[4] = 1 , j=4, i=0, i,j距離爲4比k還大，因此爲false nums = [1,2,3,4,1] k=4; nums[0] = nums[4] = 1 , j=4, i=0, i,j距離爲4沒有比k大，因此爲true

思路

1. 使用一個map存放出現過的數字以及位置
2. 如果出現重複的數字，判斷目前位置與儲存的位置距離是否小於等於k，有的話直接回傳true
3. 陣列中沒有符合的配對，回傳false

解題

```
/**
 * @param {number[]} nums
 * @param {number} k
 * @return {boolean}
 */
var containsNearbyDuplicate = function(nums, k) {
    if(nums.length <= 1) return false;

    // 使用map儲存出現過的數字
    var map = {};

    for(var i in nums){
        var v = nums[i];

        if(map[v] && (i - map[v] <= k)){
```

```
            // 如果有出現重複的數字，判斷目前的位置-之前儲存的位置 <= k，符合
            return true;
        }
        // 將出現過的數字與其位置放入map
        map[v] = i;
    }

    // 全部跑完，沒符合條件的配對，return false
    return false;
};
```

LeetCode 1. Two Sum

LeetCode 1. Two Sum

題目

Given an array of integers, return indices of the two numbers such that they add up to a specific target.

You may assume that each input would have exactly one solution.

Example:

```
Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9,
return [0, 1].
```

翻譯

給一個裡面元素爲int的陣列，陣列中會有兩個元素加起來等於target，回傳這兩個元素的位置。

範例：
[2, 7, 11, 15], target = 9，2+7=9，因此回傳[1,2]

思路

相信大部分人跟我一樣，接觸leetCode的時候都是先寫這題，這題完全就是Easy的題目，使用雙迴圈，如果nums[i]+nums[j] = target 就回傳i,j

解題

```
/**
 * @param {number[]} nums
 * @param {number} target
 * @return {number[]}
 */
var twoSum = function(nums, target) {

    var map = {};
```

```
    for(var i = 0 ; i < nums.length ; i++){
        var v = nums[i];

        for(var j = i+1 ; j < nums.length ; j++ ){
            if(  nums[i] + nums[j]  == target ){
                return [i,j];
            }
        }

    }
};
```

進階

上面雙迴圈的時間複雜度是O(n^2)，效率明顯不太好，用map就可以在一次走訪中找到i,j的位置

```
var twoSum = function(nums, target) {

    var map = {};
    for(var i = 0 ; i < nums.length ; i++){
        var v = nums[i];

        if(map[target-v] >= 0){
            // 如果 target - v可以在map中找到值x，代表之前已經出現過值x， ta
            // 因此回傳 x的位置與目前v的位置
            return [map[target-v],i]
        } else {
            // 使用map儲存目前的數字與其位置

            map[v] = i;
        }
    }
};
```

LeetCode 189. Rotate Array

LeetCode 189. Rotate Array

題目

Rotate an array of n elements to the right by k steps.

For example, with n = 7 and k = 3, the array [1,2,3,4,5,6,7] is rotated to [5,6,7,1,2,3,4].

Note: Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem.

翻譯

給一個n值，n代表陣列中包含1~n個元素與一個整數k，將陣列裡面的元素向右旋轉k次。

範例：

```
n=7,k=3, array[1,2,3,4,5,6,7] -->  [5,6,7,1,2,3,4]
```

思路

如果一個陣列的長度爲n，向右旋轉k次，其實就跟沒旋轉一樣，所以實際上要旋轉的次數step = n%k。

向右旋轉1次，就是將陣列最後面的元素搬到最前面，使用javascript array的pop()可以取出最後一個元素， unshift(x)可以將元素插到陣列的最前面，這解法太簡單了，因此我們這邊不用這兩個方法來解題。

這邊我們用一個暫存陣列temp儲存向右旋轉的元素，需要旋轉k次temp裡面就有幾個元素。題目要求只能在一開始的nums陣列內操作 ，所以接下來就是把nums內沒被旋轉的元素搬到nums的後面，接著將temp放到nums前面。

解題

```
/**
 * @param {number[]} nums
 * @param {number} k
 * @return {void} Do not return anything, modify nums in-place instea
 */
var rotate = function(nums, k) {
```

```
    // ex. [1,2,3] k=5  與 [1,2,3] k=2 相同
    var step = k%nums.length;
    var temp = [];

    // 將向右旋轉的元素裝到temp, [1,2,3] k=2, temp = [2,3]
    for(var i = step - 1 ; i >= 0 ; i--) {
        var index = nums.length-1-i;
        temp.push(nums[index]);
    }

    for(var j = nums.length - 1; j >= 0 ; j--){
        if( j >= step){
            // 將nums內沒被旋轉的元素往後移k格，[1,2,3] -> [x,x,1]
            nums[j] = nums[j-step];
        } else {
            // 將temp放到nums的前面 [2,3,1]
            nums[j] = temp[j];
        }
    }

};
```

用**javascript**強大的**array method**解題

```
var rotate = function(nums, k) {
    var step = k%nums.length;
    for(var i = 0 ; i < step ; i++){
        var value = nums.pop();
        nums.unshift(value);
    }
}
```

LeetCode 104. Maximum Depth of Binary Tree

LeetCode 104. Maximum Depth of Binary Tree

題目

Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

翻譯

找出一個二元樹的深度。

範例：

```
Tree1 depth = 3                 Tree2 depth = 4
   A                               3
  / \                             / \
 B   C                           4   4
    / \                         /
   D   E                       2
                              /
                             3
```

##思路 這邊用遞迴求解，節點如果left或right有值，則level+1，並判斷left或right有沒有子節點，直到底層，轉換成公式如下：

```
   fn(n) = 0, if null;
   fn(n) = 1 + max(f(n.left), f(n.right))
```

以上面tree1爲例
*nodeA的深度 = 1 + max (nodeB深度 , nodeC深度) // B，C取深度較深的
*nodeB 沒有子節點，nodeB depth = 1
*nodeC 有子節點[D,E]，nodeB depth = 2
*nodeA depth = 1 + 2 = 3

解題

```
var maxDepth = function(root) {
    return find(root);
```

```
    // 遞迴函式
    function find(node){
        // 節點到底
        if(node === null){
            return 0;
        }

        var deepL = 1;
        var deepR = 1;
        // 有左節點，往下一層找
        if(node.left !== null){
            deepL += find(node.left)
        }
        // 有右節點，往下一層找
        if(node.right !== null){
            deepR += find(node.right)
        }

        // 回傳較大的深度depth，給上一層節點
        return deepL > deepR ?  deepL: deepR;
    }
};
```

LeetCode 226. Invert Binary Tree

LeetCode 226. Invert Binary Tree

題目

Invert a binary tree.

翻譯

反轉一個二元樹。

範例：
輸入

```
     4
   /   \
  2     7
 / \   / \
1   3 6   9
```

反轉

```
     4
   /   \
  7     2
 / \   / \
9   6 3   1
```

思路

第一次解這題的時候還跟遞迴不怎麼熟，一直用迴圈嘗試，結果也沒寫出來，那時候心中的不斷的OS，不是說好這是Easy的題目嗎!!!?

後來看討論發現用遞迴來解會比較好理解很多，如果你現在心中跟我有一樣的OS，那建議過幾周後再回來重寫一次。

反轉整顆樹，其實除了已經到底的節點(left)之外，每一個節點都需要把他的左右節點互換，例如上面的[4, left:2, right:7]，就要把2，7互換， [2, left:1, right:3]，[7, left:6, right:9]也都是一樣，到了底部[1,3,6,9]這幾個點就不用交換，表示遞迴結束。

解題

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *     this.val = val;
 *     this.left = this.right = null;
 * }
 */
/**
 * @param {TreeNode} root
 * @return {TreeNode}
 */
var invertTree = function(root) {
    //節點爲null或沒有子節點，不用反轉，終止遞迴
    if(root === null || (root.right === null && root.left === null)){
        return root;
    }
    // 左節點爲本來的右節點反轉，右節點爲本來的左節點反轉
    var temp = root.left;
    root.left = invertTree(root.right);
    root.right = invertTree(temp);

    return root;

};
```

```
* 以[4][2,7][1,3,6,9]這個樹來說明
* 執行時傳入節點4，發現有左節點[2]，右節點[7]，將[2][7]交換
* 交換同時發現新的左節點[7]有子節點[9,6]，交換成爲[6,9]，因爲[6][9]都沒子節點
* 接著看新的右節點[2]，有子節點[1,3]，步驟同上
* 上述步驟執行後就會得到反轉的tree [4][7,2][9,6,3,1]
```

LeetCode 100. Same Tree

LeetCode 100. Same Tree

題目

Given two binary trees, write a function to check if they are equal or not.

Two binary trees are considered equal if they are structurally identical and the nodes have the same value.

翻譯

給定兩個二元樹，判斷這兩個樹的每一個節點中的值與節點位置是否都相同

思路

這題跟LeetCode 226. Invert Binary Tree很像，用遞迴解會比迴圈容易理解很多。

比較兩個節點的值(val)，如果val不同或是子節點數量不相同，都是false，子節點不同就是說其中一邊有左右節點，另外一邊就只有一個左節點或右節點這種情況。 當目前val與子節點數量都相同，再比較子節點是否相同，將子節點丟入判斷是否相同的function，直到比較完每個節點都是一樣的，結果才會是true。

解題

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *     this.val = val;
 *     this.left = this.right = null;
 * }
 */
/**
 * @param {TreeNode} p
 * @param {TreeNode} q
 * @return {boolean}
 */
var isSameTree = function(p, q) {
    // 如果兩個node都是null，代表所有node比較都是一樣的
    if(p === null && q=== null){
        return true;
```

```
    }

    // one null, other is not null, false
    if(p !== null && q=== null || p === null && q !== null){
        return false;
    }

    // val diff, false
    if(p.val != q.val){
        return false;
    }
    // find next level of tree
    return isSameTree(p.right,q.right)&&isSameTree(p.left, q.left);
};
```

以treeA[1,2,0,3,4,null,5]與treeB[1,2,0,3,4,5,5]兩個二元樹爲例

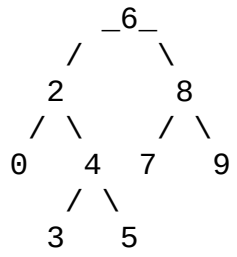
- treeA與treeB第一個節點的val均爲1，也都有兩個子節點，進行子節點的比較
- 比較treeA 下一層的左節點2與 treeB 下一層的左節點2，兩者val相同
- 再向下比較，會發現treeA，treeB在左節點下的[3,4]也都是相同的
- 接著比較treeA 下一層的右節點0與 treeB 下一層的右節點0
- treeA節點0下一層的左節點爲null，右節點爲5，treeB節點0下一層的左節點則是5
- 以上可以判斷這兩個tree是不相等的

LeetCode 235. Lowest Common Ancestor of a Binary Search

LeetCode 235. Lowest Common Ancestor of a Binary Search

Given a binary search tree (BST), find the lowest common ancestor (LCA) of two given nodes in the BST.

According to the definition of LCA on Wikipedia: “The lowest common ancestor is defined between two nodes v and w as the lowest node in T that has both v and w as descendants (where we allow a node to be a descendant of itself).”



For example, the lowest common ancestor (LCA) of nodes 2 and 8 is 6. Another example is LCA of nodes 2 and 4 is 2, since a node can be a descendant of itself according to the LCA definition.

翻譯

給一個二元搜尋樹(BST)，再給兩個樹中的節點Node，找出這兩個Node的最低共同祖先節點，根據LCA在維基百科的定義: "最低共同祖先節點，是指在一個Tree T中的Node v, Node w存在一個最低層級共同的祖先T，同時 v與w也可以算是自己的祖先節點"

範例：以上面的樹來看，[2,8]的最低共同祖先節點爲6。
[2,4]的最低共同祖先節點爲2，因爲2也算是自己的祖先節點

思路

一個BST，若任意節點的左子樹不爲空，則左子樹所有節點的值都會比根節點小，從上面的樹可以看到6的左子樹裡面的節點爲[2,0,4,3,5]均比6小。

1. 取得根節點root的值，如果值落在目標節點[p,q]之間，代表root就是最低共同祖先節點
2. 如果root比[p,q]都還小，用root.right取代root繼續往下找
3. 如果root比[p,q]都還小，用root.left取代root繼續往下找

解題

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *     this.val = val;
 *     this.left = this.right = null;
 * }
 */
/**
 * @param {TreeNode} root
 * @param {TreeNode} p
 * @param {TreeNode} q
 * @return {TreeNode}
 */
var lowestCommonAncestor = function(root, p, q) {
    var count = 0;

    while(true){
        var value =  root.val;

        if(p.val >= value && value >= q.val || p.val <= value && valu
            return root;
        } else if(p.val > value && q.val > value){
            root =  root.right;
        } else {
            root =  root.left;
        }
    }
};
```

LeetCode 102. Binary Tree Level Order Traversal

LeetCode 102. Binary Tree Level Order Traversal

題目

Given a binary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

For example: Given binary tree [3,9,20,null,null,15,7],

翻譯

給一個二元樹，每一個階層從左到右裝成一個列陣，再依序放到一個大的列陣裡面。

範例：

```
[3,9,20,null,null,15,7]
   3
  / \
 9   20
     / \
    15 17
```

回傳

```
[
  [3],
  [9,20],
  [15,7]
]
```

思路

107. Binary Tree Level Order Traversal II 跟這題幾乎一樣，只是107要求的是下面階層的清單放在上面。這邊我們改用迴圈來解，步驟如下：

1. 用一個map來存放當前節點node的深度level與值val
2. 將node放到一個watinglist陣列裡面，一開始level爲0
3. 取出watinglist內的node，如果map裡面已經有該level的陣列，增加到裡面，如果沒有，新增map[level]的新陣列
4. 如果node有子節點，level+1後放入watinglist裡面等處理

5. 處理watinglist下一個node值到沒有任何元素放在watinglist中

解題

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *     this.val = val;
 *     this.left = this.right = null;
 * }
 */
/**
 * @param {TreeNode} root
 * @return {number[][]}
 */


var levelOrder = function(root) {
    if(root === null || root.length === 0){
        return [];
    }

    // 使用map[level]的list儲存相同level節點的值
    var map = {};

    // 待處理節點的等待陣列，第一個要處理的當然就是root
    var waitlist = [ { level:0, node:root} ];

    // 處理到沒有待處理
    while(list.length > 0){

        // 取出waitlist最後一個node來處理
        var cur = waitlist.pop();
        var node = cur.node;

        // 如果當前node沒有其他同level的node，產生一個該level的list來裝
        if(!map[cur.level]){
            map[cur.level] = [node.val];
        } else {
            map[cur.level].push(node.val);
        }

        // 有子節點放入待處理清單，注意right要先放，因爲上面是抓最後一個放入的
        if(node.right){
            waitlist.push({level:cur.level+1, node:node.right});
        }

        if(node.left){
            waitlist.push({level:cur.level+1, node:node.left});
        }

    }
    var result = [];
```

```
    for(var i in map){
        result.push(map[i]);
    }

    return result;
};
```

以上面的 [3,9,20,null,null,15,7]來看

1. 第一次執行時會將整個(node -> level:0，val:3)放入waitlist
2. 迴圈開始，抓出level:0，val:3的節點，發現map[0]沒有東西，因此map[0] = [3]
3. [3]的right[20]，level+1後放入waitlist，left[9]level+1後放入waitlist
4. 迴圈，取出最後放入的節點[9]，map[1] = [9]，[9]沒有子節點
5. 迴圈，取出剩下的[20]，map[1] = [9,20]，waitlist放入right[7], left[15]
6. 接下來迴圈繼續處理[15],[7]

LeetCode 107. Binary Tree Level Order Traversal II

LeetCode 107. Binary Tree Level Order Traversal II

題目

Given a binary tree, return the bottom-up level order traversal of its nodes' values. (ie, from left to right, level by level from leaf to root).

For example: Given binary tree [3,9,20,null,null,15,7],

翻譯

給一個二元樹，每一個階層從左到右裝成一個list，再從下到上放到一個大list裡面。

範例：
[3,9,20,null,null,15,7]

```
   3
  / \
 9   20
     / \
    15 17
```

回傳

```
[
  [15,7],
  [9,20],
  [3]
]
```

思路

1. 可以先去寫102. Binary Tree Level Order Traversal這題，兩題是一樣的，這題要再把102題最後的陣列反轉。這邊我們不用迴圈，改用遞迴來寫。
2. 先處理level 0 的root，將root做爲第一個節點傳入遞迴function，當節點沒有left也沒有right的時候，停止遞迴。
3. 根據傳入的level，如果map裡面已經有該level的list，增加到裡面，如果沒有，map[level]

= [node.val]。
4. 如果當前節點有子節點，先處理left，再處理right。
5. 不斷的重複3,4步驟值到沒有子節點。

解題

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *     this.val = val;
 *     this.left = this.right = null;
 * }
 */
/**
 * @param {TreeNode} root
 * @return {number[][]}
 */
var levelOrderBottom = function(root) {
    if(root == null) return [];
    var result = [];

    // 使用map[level]的list儲存相同level節點的值
    var map = {};

    // 傳入root，level 0
    find(root,0);

    for(var i in map){
        result.push(map[i])
    }
    return result.reverse();

    // 傳入node與level
    function find(node,level){
        if(node == null) return;
        // map[level]沒東西，map[level] = [node.val]
        if(!map[level]){
            map[level] = [node.val];
        } else {
            map[level].push(node.val);
        }

        // 先處理left，再處理right
        find(node.left,level+1);
        find(node.right,level+1);
    }


};
```

LeetCode 101. Symmetric Tree

LeetCode 101. Symmetric Tree

題目

Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).

For example, this binary tree [1,2,2,3,4,4,3] is symmetric: But the following [1,2,2,null,3,null,3] is not:

翻譯

給一個二元樹，判斷他是不是鏡像樹(從root爲中心，左邊跟右邊爲鏡像)。

範例：
[1,2,2,3,4,4,3]是鏡像樹

```
     1
   /   \
  2     2
 / \   / \
3   4 4   3
```

[1,2,2,null,3,null,3]就不是鏡像樹

```
     1
   /   \
  2     2
   \     \
    3     3
```

思路

判斷右邊是不是左邊的鏡像，就是先把右邊的樹反轉，再判斷是否與左邊的樹相等。 剛好怎麼把樹反轉在Invert Binary Tree有寫過，Same Tree也寫過，把這兩個組合起來就是本題的解了

解題

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *     this.val = val;
```

```
 *     this.left = this.right = null;
 * }
 */
/**
 * @param {TreeNode} root
 * @return {boolean}
 */
var isSymmetric = function(root) {
    if(root == null || (root.right == null && root.left == null) ){
        return true;
    }

    // 先將right反轉，再比對是否相等
    root.right = revertTree(root.right);
    return isSameTree(root.left,root.right);

    // 反轉樹
    function revertTree(node){
        if(node == null || node.left == null && node.right == null){
            return node;
        }
        var temp = revertTree(node.left);
        node.left = revertTree(node.right);
        node.right = temp;
        return node;
    }


    // 比對是否相等
    function isSameTree(left,right){
        if(left == null && right== null){
            return true;
        }

        if(left == null && right != null || right == null &&left != r
            return false;
        }

        if(left.val != right.val){
            return false;
        }

        return isSameTree(left.right, right.right) && isSameTree(left
    }
};
```

LeetCode 110. Balanced Binary Tree

LeetCode 110. Balanced Binary Tree

題目

Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of every node never differ by more than 1.

翻譯

給一個二元樹，判斷這是不是一個高度平衡的樹。

在這個問題中，高度平衡樹的定義是一個樹兩個子樹每個node的level不能差超過1。

範例：

高度平衡樹

```
      1
    /   \
   2     5
  / \     \
 3   4     6
```

在節點5的時候，節點5左子樹`level = 1`，右子樹`level = 3`， `3-1 > 1` 因此這是非高度

```
      1
    /   \
   2     5
  / \     \
 3   4     6
             \
              7
```

思路

1. 這題會用到之前寫過的LeetCode 104. Maximum Depth of Binary Tree
2. 尋找node左樹的深度與右樹的深樹後相減，如果差超過1，表示非高度平衡樹
3. 如果沒差超過1，傳入左節點與右節點繼續判斷是否爲高度平衡樹

解題

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *     this.val = val;
 *     this.left = this.right = null;
 * }
 */
/**
 * @param {TreeNode} root
 * @return {boolean}
 */
var isBalanced = function(root) {
    if(root == null || (root.right == null && root.left == null)) re

    // 找出左樹，右樹的深度
    var dL = findDeep(root.left);
    var dR = findDeep(root.right);

    // 深度是否差1內
    var diff = Math.abs(dL-dR) <= 1;

    //如果深度差超過1, return false
    //深度沒差超過1，傳入左右節點繼續判斷
    return diff && isBalanced(root.left) && isBalanced(root.right);

};

// 找出節點深度
function findDeep(root){
    if(root == null) {
        return 0;
    }
    var deepL = 1+findDeep(root.left);
    var deepR = 1+findDeep(root.right);

    return deepL > deepR ? deepL:deepR;
}
```

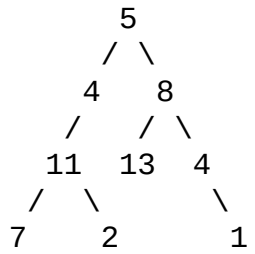
LeetCode 112. Path Sum

LeetCode 112. Path Sum

題目

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.

For example: Given the below binary tree and sum = 22,



return true, as there exist a root-to-leaf path 5->4->11->2 which sum is 22.

翻譯

給一個二元樹與另外一個數字sum，判斷二元樹從根加到葉是否有一個路徑可以得到一個與sum相同的數字。

範例：
例如上面的樹，sum = 22，可以找到 5 + 4 + 11 + 2 = 22，因此return true。

思路

1. 使用遞迴來加總每一條路徑的總和，其實就是LeetCode 104. Maximum Depth of Binary Tree的變形，在104中計算的每條路徑的深度，這邊改成計算每個條路徑的總和
2. 以上面範例的樹爲例，一開始root[5]，總和s = 0，先計算左節點[4]，這時候s=9
3. [4]沒有右節點，因此只要繼續計算左節點[11]，s = 9 + 11 =20
4. [11]左邊爲[7]，s = 27 ， 因爲[7]已經是left，所以比對 s == sum ， 不相等就計算下一條路徑
5. [5,4,11,7]這個路徑不符合，退回[5,4,11]，計算[11]的右節點[2]，s = 22，等於sum，因此return true。

解題

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *     this.val = val;
 *     this.left = this.right = null;
 * }
 */
/**
 * @param {TreeNode} root
 * @param {number} sum
 * @return {boolean}
 */
var hasPathSum = function(root, sum) {
    if(root == null) return false;

    // 如果不太清楚怎麼運作，這邊用一個list來儲存每一條root計算後的加總
    // 接開下面三行程式與function sumR2L() 裡面 list.push(s)這行單純是爲了
    // 測試資料填 [5,4,8,11,null,13,4], 1 ，可以看到加總爲[20,16,17]
    var list = [];
    sumR2L(root,0);
    console.log(list)

    // 起點爲root，這時候總和s=0
    return sumR2L(root,0);

    // 遞迴function，計算到目前節點的總和，到leaf的時候停下並且判斷是否與傳入的
    function sumR2L(root,s){
        // 到底的時候，判斷總和是否與sum相同
        if(root.left == null && root.right == null){
            list.push(s);   //將root -> leaft 的加總放到list，測試用

            s += root.val;
            return s == sum;
        }

        // 左邊還沒到底，右邊已經到底，繼續計算左樹的總和
        if(root.left != null && root.right == null){
            return sumR2L(root.left, s+root.val);
        }
         // 右邊還沒到底，左邊已經到底，繼續計算右邊的總和
        if(root.right != null && root.left == null){
            return sumR2L(root.right, s+root.val);
        }

        // 兩邊的樹都還沒到底，繼續分別計算總和，因爲只要有一條路徑成立就是true
        return sumR2L(root.left, s+root.val)||sumR2L(root.right, s+r
    }
};
```

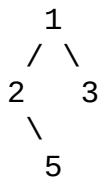
LeetCode 257. Binary Tree Paths

LeetCode 257. Binary Tree Paths

題目

Given a binary tree, return all root-to-leaf paths.

For example, given the following binary tree:



All root-to-leaf paths are: ["1->2->5", "1->3"]

翻譯

給一個二元樹，回傳全部root到leaf的路徑，例如圖中的二元樹，
有兩條路徑["1->2->5", "1->3"]

思路

跟111. Minimum Depth of Binary Tree差不多，題目111中我們已經能找出每條path的深度，這邊改成紀錄路徑，走訪每一條路徑到達leaf的時候，就把路徑放到陣中。

解題

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *     this.val = val;
 *     this.left = this.right = null;
 * }
 */
/**
 * @param {TreeNode} root
 * @return {string[]}
 */
var binaryTreePaths = function(root) {
```

```
    if(!root) return [];
    // 使用list儲存每一條path
    var list = [];
    findPath(root,"");
    return list;

    // 走訪每一條path
    function findPath(node,str){
        // left, right 都是null，代表已經到了leaf，一條路徑完成
        if(node.right == null && node.left == null){
            list.push(str + node.val); // 走完一條路徑放到list
        } else {
            // 將目前節點的值加到str，並繼續走訪子節點
            if(node.left != null) {
                findPath(node.left,str + node.val + "->" );
            }
            if(node.right != null) {
                findPath(node.right,str + node.val + "->" );
            }
        }
    }
};
```

LeetCode 206. Reverse Linked List

LeetCode 206. Reverse Linked List

題目

Reverse a singly linked list.

翻譯

反轉一個連結串列。

範例： { val:1, next: {val:2, next:null} } }

反轉： { val:2, next: {val:1, next:null} } }

思路

反轉一個連結陣列，需要一個指標來操作連結陣列，找出指標前的陣列並且將這個陣列加到目前指標節點的位置，這時候指標向前。

例如連結head[3,2,1]

1. 指標一開始放在[2]的位置，第一步將指標前的[3]從連結中切出來然後加到[2]的後面，連結陣列變成[3,2,1]。
2. 接下來指標向後移動到[1]，切出前面的[2,3]應且加到[1]之後，連結陣列變成[1,2,3]
3. 指標在往後移，發現後面已經沒有節點，交換完成。

解題

```
/**
 * Definition for singly-linked list.
 * function ListNode(val) {
 *     this.val = val;
 *     this.next = null;
 * }
 */
/**
 * @param {ListNode} head
 * @return {ListNode}
 */
var reverseList = function(head) {
```

```
    if(!head){
        return null;
    }

    if(!head.next){
        return head;
    }

    var prev = head;
    var cur  = head.next;
    prev.next = null;

    while(cur != null){
        var temp = cur;   // 用temp來操作目前的node
        cur = cur.next;   // 目前的node指標先往下，不然會被後面的操作影養

        // 這邊其實是兩個步驟
        // 我們只需要當前node的值，不需要他的next，temp.next = null
        // 將之前的linked list加到當前node後面  ， temp.next = prev
        temp.next = prev;

        // 交換完成的temp變成新的prev
        prev = temp;
    }
    return prev;
};
```

```
* 給一個linked list = [1,2,3,4]。 執行while前，prev = head = [1,2,3,4]
* 來看看prev，我們只需要[1]，後面的[2,3,4]都必須移除，prev.next=null，prev
* 第一次進入while， temp = cur = [2,3,4]。 這時候要讓cur指向[3,4]，不然對t
* 這邊我們需要只留下[2]而不是[2,3,4]，temp.next = null ； temp = [2]
* 將prev接到temp後面，temp.next = prev； temp = [2,1]； cur = [3,4]
* 交換完成，temp變成新的prev，prev = temp， prev = [2,1]
* 重複以上步驟，值到反轉完成。
```

LeetCode 237. Delete Node in a Linked List

LeetCode 237. Delete Node in a Linked List

題目

Write a function to delete a node (except the tail) in a singly linked list, given only access to that node.

Supposed the linked list is 1 -> 2 -> 3 -> 4 and you are given the third node with value 3, the linked list should become 1 -> 2 -> 4 after calling your function.

翻譯

刪除連結串列的一個節點。

範例：
連結陣列1 -> 2 -> 3 -> 4 ，傳入3，則執行後連節陣列變成1 -> 2 -> 4

思路

連結陣列每個節點(Node)有兩個屬性，值(val)與下一個節點(next)，刪除節點其實就是讓連結的val與next。都跳過當前節點指向下一個節點。

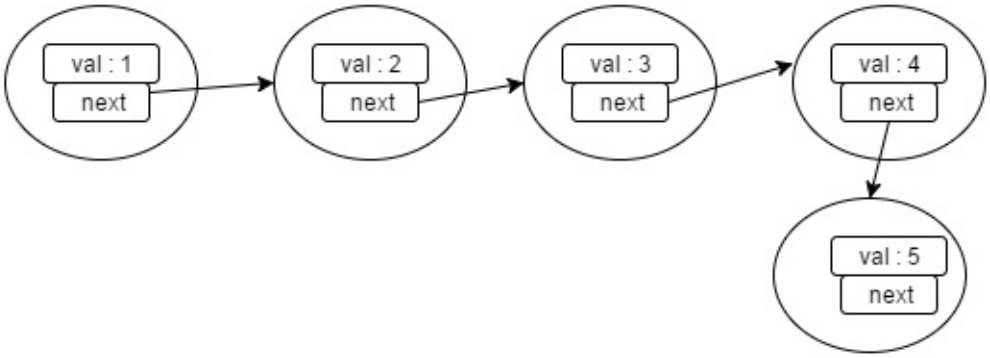
其實這題沒很好，可以跳過，沒有顯示出連結陣列的特性，203. Remove Linked List Elements會比較完整的使用到連結陣列的特性。

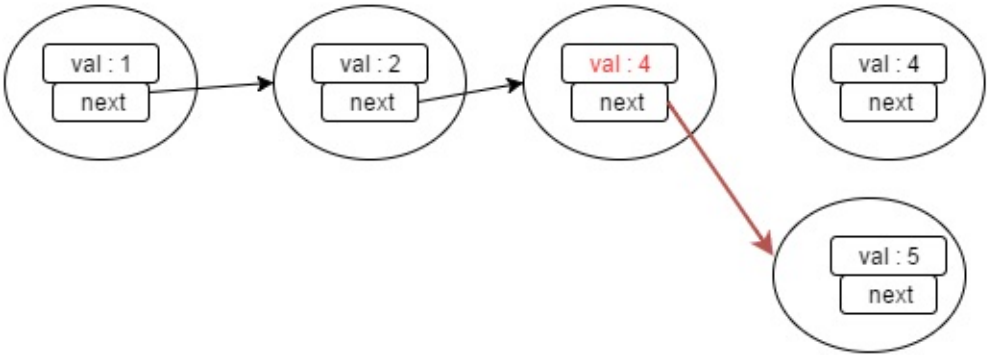
解題

```
/**
 * Definition for singly-linked list.
 * function ListNode(val) {
 *     this.val = val;
 *     this.next = null;
 * }
 */
/**
 * @param {ListNode} node
 * @return {void} Do not return anything, modify node in-place instea
 */
var deleteNode = function(node) {
    node.val = node.next.val;
```

```
    node.next = node.next.next;
};
```

以上程式碼用圖解釋，一開始的連結如上圖，將val:3節點的值用val:4取代，並將node.next指向val:5節點，本來val:4的記憶體空間就會被釋放出來





LeetCode 83. Remove Duplicates from Sorted List

LeetCode 83. Remove Duplicates from Sorted List

題目

Given a sorted linked list, delete all duplicates such that each element appear only once.

For example,
Given 1->1->2, return 1->2.
Given 1->1->2->3->3, return 1->2->3.

翻譯

改一個排序過的連結陣列，刪除重複的節點。

範例：
[1,1,2] -> return [1,2]
[1,1,2,3,3] -> return [1,2,3]

思路

因爲是排序過的，所以只要判斷後面的節點與當前的是否相同，如果後面的節點跟目前的相同，就跳過他。

解題

```
/**
 * Definition for singly-linked list.
 * function ListNode(val) {
 *     this.val = val;
 *     this.next = null;
 * }
 */
/**
 * @param {ListNode} head
 * @return {ListNode}
 */
```

```
var deleteDuplicates = function(head) {
    if(head === null || head.next ===null) return head;
    var cur = head;

    while(cur.next !== null){
        // 如果目前Node的值與下一個相同，跳過下一個
        if(cur.val == cur.next.val){
            cur.next = cur.next.next;
        } else {
            cur = cur.next;
        }
    }
    return head;
};
```

LeetCode 141. Linked List Cycle

LeetCode 141. Linked List Cycle

題目

Given a linked list, determine if it has a cycle in it.

Follow up: Can you solve it without using extra space?

翻譯

給一個連結陣列，判斷裡面是不是包含一個循環連結。

進階： 你能不使用額外的空間來解這題嗎?

範例:

- [1,2,3,2,3,2.....]，[1]的next是[2]
- [2]的next是[3]
- [3]的next是[2]，從上一步我們知道2的next是[3]，所以這是一個循環連結

假如[1,2,3,2,3]，那就不是一個連結陣列迴圈，因爲最後的[3]next並不是[2]，所以循環連結不成立。

思路

拜訪節點後就給他一個拜訪過的標計，如果目前拜訪的節點已經被標計了，代表這是一個循環連結。

解題

```
/**
 * Definition for singly-linked list.
 * function ListNode(val) {
 *     this.val = val;
 *     this.next = null;
 * }
 */
/**
 * @param {ListNode} head
 * @return {boolean}
```

```
 */
var hasCycle = function(head) {
    if(head == null || head.next == null){
        return false;
    }
    var node = head;
    while(node != null ){
        if(node.flag){
            return true;
        }
        // 標記這個node已經跑過
        node.flag = true;
        node = node.next;
    }
    return false;
```

進階

- 使用快慢指針這個想法來解
- 慢指針一次只跑一格，快指針一次跑兩格
- 如果是一個循環連結，快慢指針一定會相遇

```
/**
 * @param {ListNode} head
 * @return {boolean}
 */
var hasCycle = function(head) {
    if(head == null || head.next == null){
        return false;
    }

    var slow = head.next;
    var fast = head.next.next;
    if(fast == null){
        return false;
    }

    while(slow != fast){
        if(fast.next == null){
            return false;
        }
        slow = slow.next;
        fast = fast.next.next;

        if(slow == null || fast == null){
            return false;
        }
    }
    return true;
}
```

LeetCode 21. Merge Two Sorted Lists

LeetCode 21. Merge Two Sorted Lists

題目

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

翻譯

融合兩個排序過的連結串列爲一個新的連結串列後回傳。

範例：
[1,2,2,3] + [1,3] = [1,1,2,2,3,3]

思路

讀取L1目前的值與L2目前的值比較，如果L1.val < L2.val，將當前的L1節點加入新的連結串列(result)，然後L1指向下一個節點。
如果L1.val > L2.val較小，則把L2當前的節點加到result，直到L1或L2一方爲null則停止比較，並且將另外一邊剩下的節點加入result。

解題

```
/**
 * Definition for singly-linked list.
 * function ListNode(val) {
 *     this.val = val;
 *     this.next = null;
 * }
 */
/**
 * @param {ListNode} l1
 * @param {ListNode} l2
 * @return {ListNode}
 */
var mergeTwoLists = function(l1, l2) {
    // 儲存結果的ListNode
    var result = new ListNode(0);
    // 目前Node位置
    var c = result;
```

```
    while(l1 !== null && l2 !==null){
        // l1,l2較小的數加入result
        if(l1.val > l2.val){
            c.next = l2;
            l2 = l2.next;
        } else {
            c.next = l1;
            l1 = l1.next
        }
        c = c.next;
    }

    //將l1,l2剩下的Node加到result
    if(l1 !== null){
        c.next = l1;
    }

    if(l2 !== null){
        c.next = l2;
    }
    return result.next
};
```

LeetCode 24. Swap Nodes in Pairs

LeetCode 24. Swap Nodes in Pairs

題目

Given a linked list, swap every two adjacent nodes and return its head.

For example, Given 1->2->3->4, you should return the list as 2->1->4->3.

Your algorithm should use only constant space. You may not modify the values in the list, only nodes itself can be changed.

翻譯

給一個連結陣列，交換兩兩相鄰的節點並且回傳。

範例： 1->2->3->4, return 2->1->4->3。

你的演算法不能改變節點裡面的值，只能把節點搬來搬去。

思路一

先不管最後一條規則，也就是直接改變節點裡面值

1. 用prev表示前面的節點，cur表示後面的節點，跟一般交換的寫法一樣，先用一個temp儲存prev的值
2. 將cur的值塞給prev，然後再將temp的值塞給cur就完成交換
3. prev與cur尋找下一對node，繼續交換到結束

解題

改變node內的值

```
/**
 * Definition for singly-linked list.
 * function ListNode(val) {
 *     this.val = val;
 *     this.next = null;
 * }
 */
```

```
/**
 * @param {ListNode} head
 * @return {ListNode}
 */
var swapPairs = function(head) {
    if(head == null || head.next == null) return head;
    var prev = head;
    var cur  = head.next;

    while(prev != null && cur != null){
        // 很一般的交換
        var temp = prev.val;
        prev.val = cur.val;
        cur.val = temp;

        if(cur.next == null || cur.next.next == null) break;
        // 處理下一對node
        prev = cur.next;
        cur  = cur.next.next;
    }
    return head;
};
```

思路二

1. 題目要求不能改變節點裡面的值，所以node.val這種寫法應該是不行的
2. 首先先弄一個不動的首節點firstNode，後面節點移來移去時才不會受影響
3. 以[1,2,3,4]，一開始要交換的爲[1.2]，因此需要先儲存[3,4]稍後再處理(下圖1)
4. 儲存後將list要處理的部分[1,2]跟不處理的部分[3,4]切開 (下圖2)，切開的同時讓[2]的next指向[1]
5. 將前一個節點prev的next指向[2]，因爲[2]的next已經指向[1]，因此這邊已經完成[1,2] -> [2,1]的步驟
6. 接下來把之前儲存的[3,4]接到[1]後面，就可以繼續處理[3,4] ```

不改動node內的值

var swapPairs = function(head) { if(head == null || head.next == null) return head;

```
var firstNode = new ListNode(0);

firstNode.next = head;
var cur = head;
var prev = firstNode;
// 圖1的部分，目前處理的節點cur，前一個節點prev

var nextKeep; // 預備用來儲存後面未處理的節點

while (cur!=null && cur.next!=null){
    nextKeep = cur.next.next; //圖1，相將後面的節點[3,4]儲存起來
    cur.next.next = cur; //圖2，將[1,2]與[3,4]斷開，這時候[2]的next已經變
    // 圖3將prev的next指向[2]，注意這時候[2]的next已經指向[1]，整理一下其實女
    prev.next = cur.next;
```

```
    //圖5，將[3,4]接回[1]
    cur.next = nextKeep;

    //處理下一組
    prev = cur;
    cur = cur.next;
}
return firstNode.next;

}```
```

圖解不改動**node**值的寫法

figure 1

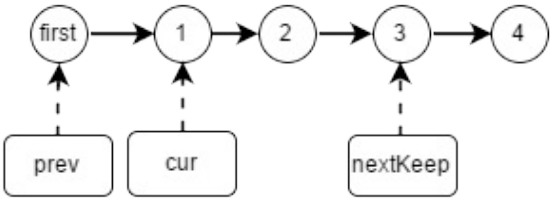


figure 2

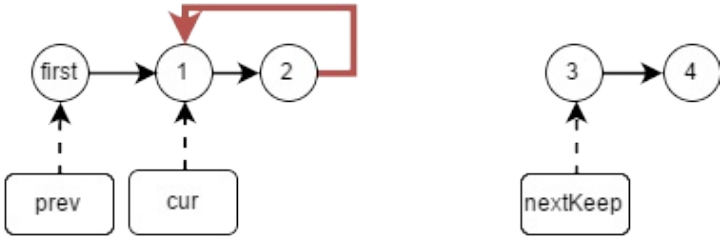


figure 3

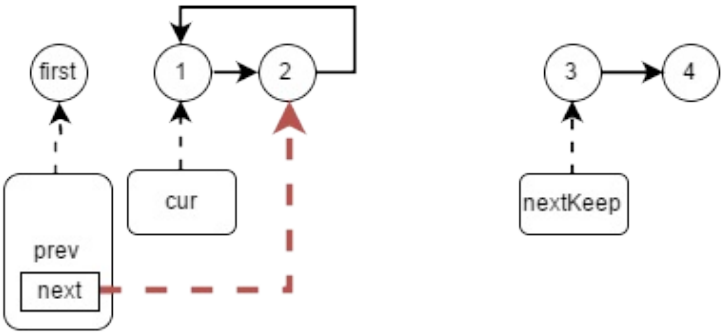


figure 4

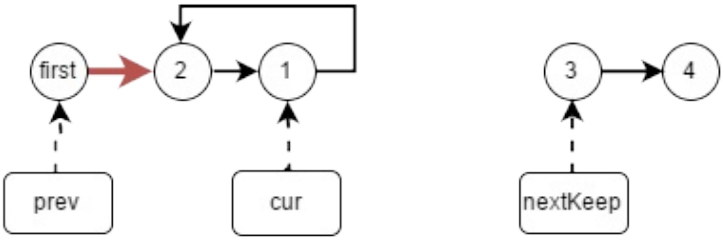


figure 5

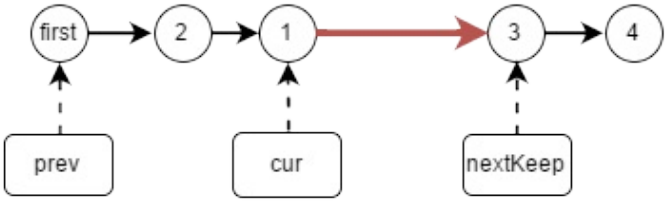
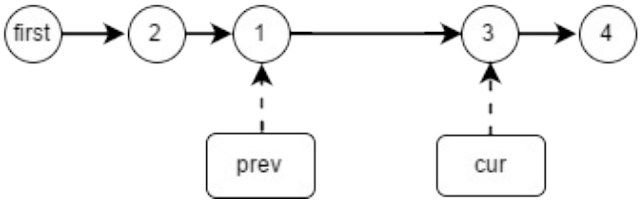


figure 6



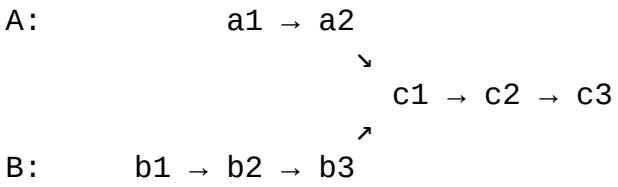
LeetCode 160. Intersection of Two Linked Lists

LeetCode 160. Intersection of Two Linked Lists

題目

Write a program to find the node at which the intersection of two singly linked lists begins.

For example, the following two linked lists:



begin to intersect at node c1.

Notes:

- If the two linked lists have no intersection at all, return null.
- The linked lists must retain their original structure after the function returns.
- You may assume there are no cycles anywhere in the entire linked structure.
- Your code should preferably run in O(n) time and use only O(1) memory.

翻譯

寫一支程式找出兩個連結陣列的第一個交集的節點。

注意：

- 如果兩個連結陣列沒有交集，回傳null
- 連結陣列必須保持他們的原始結構
- 可以假設不會出現迴圈連結陣列
- 程式只能在O(n)的時間跑完而且只能使用O(1)的空間

思路

先判斷A跟B哪個連結陣列長度比較長，接著移除較長那邊前面的節點，接下來就兩兩節點比較是否相等。

解題

```
/**
 * Definition for singly-linked list.
 * function ListNode(val) {
 *     this.val = val;
 *     this.next = null;
 * }
 */

/**
 * @param {ListNode} headA
 * @param {ListNode} headB
 * @return {ListNode}
 */
var getIntersectionNode = function(headA, headB) {
    // 判斷headA, headB 哪個比較長
    var diff = getDiff(headA, headB);

    // headA比較長，跳過前面多出的部分
    if(diff > 0){
        while(diff > 0){
            headA = headA.next;
            diff--;
        }
    } else {
        // headB比較長，跳過前面多出的部分
         while(diff < 0){
            headB = headB.next;
            diff++;
        }
    }

    // 每個節點進行比較
    while(headA !=null){
        if(headA == headB){
            return headA;
        }
        headA = headA.next;
        headB = headB.next;
    }

    return null;


    // 取得兩個linked list長度差異
    function getDiff(nodeA,nodeB){
        var aLength = 0;
        var bLength = 0;

        while(nodeA != null ){
            aLength++;
            nodeA = nodeA.next;
```

```
        }
        while(nodeB != null ){
            bLength++;
            nodeB = nodeB.next;
        }

        return aLength - bLength;
    }
};
```

LeetCode 19. Remove Nth Node From End of List

LeetCode 7. Reverse Integer

題目

Given a linked list, remove the nth node from the end of list and return its head.

For example,

```
Given linked list: 1->2->3->4->5, and n = 2.

After removing the second node from the end, the linked list becor
```

Note:

Given n will always be valid. Try to do this in one pass.

翻譯

給一個連結陣列，移除連結陣列從後面數來第n個節點

例如，

```
1->2->3->4->5, n = 2.
```

從後面數2個，移除4後的連結陣列變成 `1->2->3->5.`

注意:

n不會比連結陣列還長，試著跑一次迴圈解題。

思路

這題可以用快慢指針的做法，就是快指針先走n個節點，慢指針再開始與快指針一起跑，當快指針到底的時候，慢指針會停在要移除的節點n上面。

不過慢指針應該要停在要移除節點的前一點才對，因爲連結陣列的刪除，應該是前一點跳過下一點，這樣當前的節點就會被刪除。 因此將慢指針的起點改在一個假的首節點0上面修正這

個問題。

解題

```
/**
 * Definition for singly-linked list.
 * function ListNode(val) {
 *     this.val = val;
 *     this.next = null;
 * }
 */
/**
 * @param {ListNode} head
 * @param {number} n
 * @return {ListNode}
 */
var removeNthFromEnd = function(head, n) {
    if(head == null) return head;

    // 用假節點當首節點，方便操作
    var node = new ListNode(0);
    node.next = head;


    var slow = node;
    var fast = head;

    // 快指針先跑n個節點
    while(n > 0){
        fast = fast.next;
        n--;
    }

    // 因爲快指針已經先跑n點，所以當快指針fast跑完，慢指針slow會在要移除的前一點
    while(fast){
        fast = fast.next;
        slow = slow.next;
    }
    if(slow.next == null){
        slow.next = null ;
    } else {
        slow.next = slow.next.next;
    }

    return node.next;
};
```

- [1,2,3,4] ， n = 2，fast = [1,2,3,4] ， slow = [0,1,2,3,4] (0爲假節點)
- n = 2，快指針先走2點，fast = [3,4]，slow = [0,1,2,3,4]
- 快慢指針一起跑，當fast = null，slow = [2,3,4]，這時候慢指針必須跳過他第2個點，也就是跳過3

LeetCode 234. Palindrome Linked List

LeetCode 234. Palindrome Linked List

題目

Given a singly linked list, determine if it is a palindrome.

Follow up: Could you do it in O(n) time and O(1) space?

翻譯

給一個單向的連結陣列，判斷他是不是一個回文連結陣列。

進階：你能使用O(n)的時間與O(1)的空間來處理這個題目嗎?

思路

1. 不考慮進階題的情況很簡單，走訪連結陣列，使用兩個字串來處理
2. 一個正向字串(str = str + value)，另外一個爲反向字串(str = value+str)
3. 最後判斷兩個字串是否相等

解題

```
/**
 * Definition for singly-linked list.
 * function ListNode(val) {
 *     this.val = val;
 *     this.next = null;
 * }
 */
/**
 * @param {ListNode} head
 * @return {boolean}
 */
var isPalindrome = function(head) {

    var rec = "";    //反向字串
    var seq = "";   //正向字串

    while(head != null){
```

```
        seq += head.val;
        rec = head.val + rec;
        head = head.next;
    }
    // 反向字串與正向字串相等就是回文陣列
    return seq == rec;

};
```

進階

因爲用了另外一個字串，因此額外空間爲O(n)，不符合進階的要求，爲了不使用額外的O(n)空間，我們先用快慢指針找出linked list的中點
，找到後從中點之後將linked list反轉再與本來的head前半段比較是否相等，這邊需要一個額外的空間儲存反轉後的linked list。

```
var isPalindrome = function(head) {

    var middle = findMiddle(head);   // 找尋中點
    var rNode = reverseNode(middle); // 從中點反轉

    // 比對反轉後的node與head前半段是否相等
    while(rNode != null){
        if(head.val != rNode.val) {
            return false;
        }
        head = head.next;
        rNode = rNode.next;
    }
    return true;



    // 使用快慢指針找出中點
    function findMiddle(node){
        var fast = node;
        var slow = node;

        while(fast != null && fast.next != null){
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }

    // 反轉linked list
    function reverseNode(node){
        if(node==null || node.next==null) return node;
        var prev = null;
        var cur  = node;
        while(cur != null){
            var temp = cur;
            cur = cur.next;
```

```
            temp.next = prev;
            prev = temp;
        }
        return prev;
    }
};
```

LeetCode 203. Remove Linked List Elements

LeetCode 203. Remove Linked List Elements

題目

Remove all elements from a linked list of integers that have value val.

Example Given: 1 --> 2 --> 6 --> 3 --> 4 --> 5 --> 6, val = 6 Return: 1 --> 2 --> 3 --> 4 --> 5

翻譯

給一個值val，移除連結陣列中所有值爲val的節點。

思路

1. 走訪連結陣列，只要比對目前節點與val相等，就移除目前這點
2. 移除一個節點，其實是讓前一個節點跳過目前節點直接指向下一個節點
3. [1,6,3,5] val=3 ，移除[3]，因此我們需要知道[3]的前一點是[6]讓[6].next -> [5]

解題

```
/**
 * Definition for singly-linked list.
 * function ListNode(val) {
 *     this.val = val;
 *     this.next = null;
 * }
 */
/**
 * @param {ListNode} head
 * @param {number} val
 * @return {ListNode}
 */
var removeElements = function(head, val) {
    if(head == null) return null;

    // prev一開始爲假節點，例如[2]，val=2的時候，就可以讓假節點指向null，完成
    var node = new ListNode(0);

    // [1,2,3] val=2，當走到[2]的時候，移除目前[2]這點的方法是讓[1]跳過2直接
    // 所以這邊需要先儲存前一個節點prev來備用
```

```
    var prev =node;
    node.next = head;

    while(head != null){
        if(head.val != val){
            // 目前節點與val不相等，往下一個
            prev = head;
            head = head.next;
        } else {
            // 目前節點與val相等，跳過目前節點
            prev.next = head.next;
            head = head.next;
        }

    }

    return node.next;

};
```

LeetCode 2. Add Two Numbers

LeetCode 2. Add Two Numbers

題目

You are given two linked lists representing two non-negative numbers. The digits are stored in reverse order
and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4) Output: 7 -> 0 -> 8

翻譯

有兩個連結陣列分別代表兩個非負整數，他們的位數是反向儲存(越前面的節點位數越低)，每一個節點代表一個位數，將這兩個連結陣列加總後以連結陣列形式回傳。

範例：
Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 個位數爲2與5，相加爲7 ； 十位數爲4+6 = 10，需要進位 ； 百位數爲3 + 4 + 1(進位) = 8，結果爲 7->0->8

思路

1. 就只是用一個新的linked list來儲存相加後的結果
2. 要注意的就是list1跟list2長度可能不一樣
3. 另外就是相加後可能比9還大，需要考慮進位的情況

解題

```
/**
 * Definition for singly-linked list.
 * function ListNode(val) {
 *     this.val = val;
 *     this.next = null;
 * }
 */
/**
 * @param {ListNode} l1
 * @param {ListNode} l2
 * @return {ListNode}
```

```
 */
var addTwoNumbers = function(l1, l2) {
    var list = new ListNode(0); //儲存輸出的結果，因爲list的指針要不斷往後
    var result = list; // 使用一個ListNode來儲存相加的結果

    var sum,carry = 0; // carry用來處理進位

    //當 list1, list2 都沒有值，而且carry也爲0的時候才結束迴圈
    while(l1 || l2 || carry > 0){
        sum = 0;

        // list1與list2長度可能不同，分開處理
        if(l1!== null){
            sum += l1.val;
            l1 = l1.next;
        }

        if(l2!==null){
            sum += l2.val;
            l2 = l2.next;
        }

        // 如果之前有進位，carry = 1；沒有的話carry = 0
        sum += carry;
        list.next = new ListNode(sum%10); //相加如果超過9，只能留下個位數
        carry = parseInt(sum/10);

        // list指標向後
        list = list.next;
    }
    // 因爲第一個節點爲假節點，跳過
    return result.next;
}
```

用遞迴解

```
var addTwoNumbers = function(l1, l2) {

    var list = new ListNode(0);
    var result = list; // 使用一個ListNode來儲存相加的結果

    add(l1,l2,0);
    return result.next;

    function add(l1,l2,gap){
        var sum = 0;
        if(l1 === null && l2 === null && gap === 0){
            return 0;
        }

        if(l1 !== null){
            sum += l1.val;
            l1 = l1.next;
```

```
        }

        if(l2 !== null){
            sum += l2.val;
            l2 = l2.next;
        }
        sum += gap;
        list.next = new ListNode(sum%10);
        gap = parseInt(sum/10);
        list = list.next;
        add(l1,l2,gap);
    }
};
```

LeetCode 292. Nim Game

LeetCode 292. Nim Game

題目

You are playing the following Nim Game with your friend: There is a heap of stones on the table, each time one of you take turns to remove 1 to 3 stones. The one who removes the last stone will be the winner. You will take the first turn to remove the stones.

Both of you are very clever and have optimal strategies for the game. Write a function to determine whether you can win the game given the number of stones in the heap.

For example, if there are 4 stones in the heap, then you will never win the game: no matter 1, 2, or 3 stones you remove, the last stone will always be removed by your friend.

翻譯

拿石頭遊戲規則，桌上有石頭n個，兩個玩家，你先玩，每次每個玩家可以拿走1~3個石頭，拿走會後一顆石頭的人獲勝，根據以上規則與初始的石頭數量，判斷你是否能贏得這個拿石頭遊戲。

例如說，如果桌上有三顆石頭，你一次全拿，獲勝。
如果桌上有4顆石頭，不管你拿走幾顆，你的對手都會獲勝。

思路

考慮到下面幾總情況

1. 石頭數量小於等於3，直接獲勝
2. 石頭數量等於4個，輸
3. 石頭數量5個，先拿走1個，獲勝
4. 由以上可以推論(5-1=4, 6-2=4, 7-3=4) 5~7都可以獲勝
5. 石頭數量8個，不管怎樣都會輸

以上可以得到結論，只要一開始的石頭數量爲4的倍數，就會輸。

解題

```
var canWinNim = function(n) {
    // 4個以下，全拿穩贏
    if(n < 4){
        return true;
```

```
    }
    // 4的倍數，穩輸
    return n%4 != 0;
};
```

LeetCode 70. Climbing Stairs

LeetCode 70. Climbing Stairs

題目

You are climbing a stair case. It takes n steps to reach to the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

翻譯

你正在爬一個階梯。到頂端總共需走n階。

每次你都可以選擇爬1階或2階，問有幾種不同的方法可以爬到階梯頂端

思路

1. n = 1， result = 1
2. n = 2, result = 1+1 (爬1階兩次 + 一次爬兩階)
3. n = 3, result = 1+2 (前面兩個case相加)
4. n = 4, result = 3+2 (前面兩個case相加)
5. 發現其實這題就是算費氏數列

解題

```
/**
 * @param {number} n
 * @return {number}
 */
var climbStairs = function(n) {
    if(n<=1){
        return 1;
    }

    var prev = 1;
    var cur  = 1;
    // 費氏數列 f(n) = f(n-1) + f(n-2)
    for(var i = 2 ; i <=n ; i++){
        var temp = cur;
        cur = cur + prev;
        prev = temp;
```

```
    }
    return cur;
};
```

LeetCode 121. Best Time to Buy and Sell Stock

LeetCode 121. Best Time to Buy and Sell Stock

題目

Say you have an array for which the i^{th} element is the price of a given stock on day i.

If you were only permitted to complete at most one transaction (ie, buy one and sell one share of the stock), design an algorithm to find the maximum profit.

Example 1: Input: [7, 1, 5, 3, 6, 4]
Output: 5

max. difference = 6-1 = 5 (not 7-1 = 6, as selling price needs to be larger than buying price)

Example 2: Input: [7, 6, 4, 3, 1]
Output: 0

In this case, no transaction is done, i.e. max profit = 0.

翻譯

有一個陣列儲存i個數字，這些元素代表一天當中股票的價格。如果在一天中只能做一次交易(買一次然後賣一次)，設計一個演算法求得最大獲利。

範例1: Input : [7, 1, 5, 3, 6, 4]
Output: 5

最大獲利爲6-1=5，因爲賣出價格必須比購入價格高，所以7-1是不可行的。

範例2: Input: [7, 6, 4, 3, 1]
Output: 0

因爲你不管今天怎麼買，賣掉都是虧錢，所以只能回傳0。

思路

1. 單一元素最大獲利 = 目前價錢 - 前面出現的最低價格
2. 跑迴圈計算每個元素可能出現的最大獲利是多少
3. 如果現在這個元素能的獲利比之前的最大獲利還大，取現在元素的獲利爲最大獲利

解題

```
/**
 * @param {number[]} prices
 * @return {number}
 */
var maxProfit = function(prices) {
    // min代表目前股票出現的最低價，一開始用MAX_SAFE_INTEGER表示無限大
    var min = Number.MAX_SAFE_INTEGER;

    // 目前獲利
    var profit = 0;

    for(var i = 0; i < prices.length ; i++){
        // 找出最低點買進
        if(prices[i] < min){
            min = prices[i];
        }

        // 計算現在的價錢可以獲利多少
        var calProfit = prices[i] - min;
        // 現在的價錢賣出是否可以獲得更高的獲利
        if(calProfit > profit ){
            profit = calProfit;
        }
    }
    return profit;
};
```

LeetCode 198. House Robber

LeetCode 198. House Robber

題目

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight without alerting the police.

翻譯

你是一個專業的小偷，今天你打算偷遍一整條街的房子，每間房子都一定有錢可以偷。不過有一個問題，就是房子之間有設警報系統，如果你連偷兩間相鄰的房子，警報系統就會引來警察，你就會被抓。

這邊會給一個list，裡面每個元素都代表這間房子內可以偷到的錢，你要怎麼安排你的偷竊計畫才能偷到最多的錢而且不會驚動警察。

範例：
[2,4,5,3]，最多可以偷到 2+5+3 = 10，因爲4+5+3 = 12雖然可以拿到比較多錢，但是會被警察抓。

思路

1. 這題跟LeetCode 121. Best Time to Buy and Sell Stock很像，我們需要計算每間房子可以偷得多少錢，因此需要一個maxS陣列才儲存偷到目前房子最多可以拿到多少錢。
2. [m0,m1,m2,m3,....]，如果房子只有一間[m0]，可以偷到的錢爲max = mo
3. 如果有房子兩棟[m0,m1]，那最多可以拿到m0,m1之中較大的錢 max = Max(m0,m1)
4. 三棟房子的情況[m0,m1,m2]，最多拿到 max = Max(m0+m2,m1);
5. 因此可以得到，目前房子n，可以拿到的錢
   max = Max(現在這棟 + 前前一可以拿到的最大金額 , 前一棟可以拿到的最大金額)

解題

```
/**
 * @param {number[]} nums
 * @return {number}
```

```
 *
 *
 *
 */
var rob = function(nums) {
    var maxS = new Array();
    if(nums.length === 0) return 0;
    if(nums.length === 1) return nums[0];
    if(nums.length === 2) return Math.max(nums[1],nums[0]);

    maxS.push(nums[0]);
    maxS.push(Math.max(nums[0],nums[1]));

    for(var i = 2 ; i < nums.length ; i++){
        //最大金額   = Max(現在金額+前前一棟最大金額 ， 前一棟最大金額)
        maxS[i] =  Math.max(nums[i] + maxS[i-2] , maxS[i-1]);;
    }
    return maxS.pop();
};
```

LeetCode 374. Guess Number Higher or Lower

LeetCode 374. Guess Number Higher or Lower

題目

We are playing the Guess Game. The game is as follows:

I pick a number from 1 to n. You have to guess which number I picked.

Every time you guess wrong, I'll tell you whether the number is higher or lower.

You call a pre-defined API guess(int num) which returns 3 possible results (-1, 1, or 0):

```
-1 : My number is lower
 1 : My number is higher
 0 : Congrats! You got it!

Example:
n = 10, I pick 6.
Return 6.
```

翻譯

給一個n值，猜一個1~n之間神秘數字x，根據猜測的結果回傳值，如果回傳-1代表神秘數字比你猜的還低，回傳1代表比你猜的還高，回傳0代表你猜對了。最後你要回傳x到底是多少。

範例：

- n = 1000，神秘數字是987 ，猜500，得到-1
- 猜 1000，得到 1
- 猜 987，得到 0， 因此最後要回傳答案 987

思路

1. 最小值爲min，最大值爲max，猜測值 g = (min+max)/2
2. 一開始min = 1， max = n ， g = (1+n) /2
3. 如果得到1， min = g
4. 如果得到-1， max = g
5. 重複以上步驟值到猜到爲止，因爲這提沒提供javascript，只好用java來解

解題

```
/* The guess API is defined in the parent class GuessGame.
   @param num, your guess
   @return -1 if my number is lower, 1 if my number is higher, otherw
      int guess(int num); */

public class Solution extends GuessGame {
    public int guessNumber(int n) {
        if(guess(n) == 0){
            return n;
        }

        int min = 1; // 最小爲1
        int max = n; // 最大爲n
        int g = min+(max-min)/2; //先猜(min+max)/2
        int result = guess(g);

        while(max > min){

            // 猜的值太小，min=g
            if(result == 1){
                min = g;
            }

            // 猜的值太大，max=g
            if(result == -1){
                max = g;
            }

            // equal with  (min+max)/2，but use  min+(max-min)/2 coul
            g = min+(max-min)/2;

            result = guess(g);
        }

        return g;

    }
}
```

LeetCode 278. First Bad Version

LeetCode 278. First Bad Version

題目

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have n versions [1, 2, ..., n] and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API bool isBadVersion(version) which will return whether version is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

翻譯

你是一個產品經理，目前正帶領一個團隊開發新產品，不幸的是，如果有一個新版的產品沒通過測試，在這版本之後全部的新版本都不會通過測試。

假如你有n個版本的產品[1,2, ...,n]，然後你想追蹤是從哪一個版本開始沒通過測試。

這邊有一個 isBadVersion(version) API可以判斷這個版本能不能通過測試。利用這個API實作一個functon來找到第一個壞掉的版本。

範例：
假如目前有5個版本，[1,2,3,4,5]，從第4個版本開始都是壞掉的，一個一個版本丟入API驗證會得到 [true,true,true,false,false]。

思路

1. 最簡單的方法就是跑迴圈，將每個版本丟入 isBadVersion中判斷，直到發現壞掉的版本，事情沒那麼美好，這種寫法會超過時間無法通過測試，因此需要另外想一個更快的演算法。
2. 以上面n=5，版本4之後就壞掉爲例，我們可以把用 isBadVersion之後的版本陣列想像成[1,1,1,0,0]，簡單說就是要找出第一個0出現的位置，其實就是猜數字之類的遊戲
3. 這邊用二分搜尋法來尋找第一個0，一開始最大值max=n，最小值min=1，搜尋的位置爲mid = (max+min)/2， 如果 isBadVersion(mid)回傳true，代表壞掉的版本在mid之後，因此min = mid+1，反之亦然，一直搜尋到min>max，mid表示找到第一個壞掉的版本

```
    example：
    [1,2,3,4,5] --> [1,1,1,0,0]
    max = 5, min = 1   -> mid = 3   isBadVersion(3) = true
    min = mid+1 = 4    -> mid = 4   isBadVersion(4) = false
    max = mid   = 4    -> max > min == false, end loop
```

解題

```
/**
 * Definition for isBadVersion()
 *
 * @param {integer} version number
 * @return {boolean} whether the version is bad
 * isBadVersion = function(version) {
 *     ...
 * };
 */

/**
 * @param {function} isBadVersion()
 * @return {function}
 */
var solution = function(isBadVersion) {

    /**
     * @param {integer} n Total versions
     * @return {integer} The first bad version
     */
    return function(n) {
        var min = 0 ;
        var max = n ;
        var mid;

        // 用二分法進行搜尋，減少loop次數
        while(max > min){
            mid = min+parseInt((max-min)/2);
            if(isBadVersion(mid)){
                max = mid;
            } else {
                min = mid+1;
            }
        }

        return max;
    };
};
```

失敗的解法

```
/**
 * @param {function} isBadVersion()
 * @return {function}
```

```
 */
var solution = function(isBadVersion) {

    /**
     * @param {integer} n Total versions
     * @return {integer} The first bad version
     */
    return function(n) {

        for(var i = 0 ; i <= n ; i++){
            if(isBadVersion(i)){
                return i;
            }
        }
    }
}
```

LeetCode 371. Sum of Two Integers

LeetCode 371. Sum of Two Integers

題目

Calculate the sum of two integers a and b, but you are not allowed to use the operator + and -.

翻譯

計算a,b的加總，但是不能用+-符號。

思路

不用加減改用位元做運算，其實我也不會阿，而且直接回傳a+b還是會通過的，如果你被這題卡住，建議先跳過。以下單純學習分享。

1. 假設輸入的數字爲a=3(0011), b=9(1001)
2. 相加不考慮進位 a = a^b = (0011)^(1001) = 1010
3. 相加只考慮進位 carry =a&b = (0011)^(1001) = 0001 進位值
4. b = 0001 << 1 = 0010 進位
5. 第一輪後 a= 10(1010), b = 2(0010)
6. 相加不考慮進位 a = a^b = (1010)^(0010) = 1000
7. 相加只考慮進位 carry =a&b = (1010)^(0010) = 0010 進位值
8. b = 0001 << 1 = 0100 進位
9. 第二輪後 a= 8(1000), b = 4(0010)
10. 相加不考慮進位 a = a^b = (1000)^(0100) = 1100
11. 相加只考慮進位 carry =a&b = (1010)^(0010) = 0000 進位值
12. b = 0000 << 1 = 0
13. b=0，相加完成， a = 1100 = 12

解題

```
var getSum = function(a, b) {
    if(b==0){return a};
    if(a==0){return b};

    while(b!=0){
        var carry = a&b; //進位值

        a = a^b;         //相加
```

```
        b = carry << 1;  //進位
    }
    return a;
};
```

LeetCode 191. Number of 1 Bits

LeetCode 191. Number of 1 Bits

題目

Write a function that takes an unsigned integer and returns the number of ’1' bits it has (also known as the Hamming weight).

For example, the 32-bit integer ’11' has binary representation 00000000000000000000000000001011, so the function should return 3.

翻譯

給一個整數，找出這個整數有幾個'1'，例如11用32-bit表示
'00000000000000000000000000001011'，總共有3個1，return 3

思路

這題不考慮效能的話只要收先將將數字n轉爲2進位字串，然後跑迴圈找出有多少1。

解題

```
/**
 * @param {number} n - a positive integer
 * @return {number}
 */
var hammingWeight = function(n) {
    var count = 0  ;

    // n轉爲二進位
    var ary = n.toString(2).split("");
    for(var i in ary){
        if(ary[i] % 2 ==1){
            count++;
        }
    }
  return count;
};
```

LeetCode 232. Implement Queue using Stacks

LeetCode 232. Implement Queue using Stacks

題目

Implement the following operations of a queue using stacks.

push(x) -- Push element x to the back of queue. pop() -- Removes the element from in front of queue. peek() -- Get the front element. empty() -- Return whether the queue is empty. Notes: You must use only standard operations of a stack -- which means only push to top, peek/pop from top, size, and is empty operations are valid. Depending on your language, stack may not be supported natively. You may simulate a stack by using a list or deque (double-ended queue), as long as you use only standard operations of a stack. You may assume that all operations are valid (for example, no pop or peek operations will be called on an empty queue).

翻譯

使用stacks來實做queue。

- push(x) -- 將一個元素x放入queue最後
- pop() -- 移除queue最前面的元素
- peek() -- 取得queue最前面的元素(不刪除)
- empty() -- 檢查queue是否是空的

注意：
你只能用標準的stack方法操作，這表示只有push到stack頂端， peek/pop stack頂端的元素，size， isEmpty這些方法是可以用的。

有些語言可能沒有支援stack，你可以使用類似的結構，例如list或deque, 只要你的操作符合上述的標準stack方法。可以假設全部的操作都是合法的(例如說當queue是empty的時候，不會執行pop這樣的操作)。

思路

用javascrip來做這題其實頗簡單的，array本身的特定就跟stack差不多

1. push --> list.push(x)的特性跟queue一樣，都是把x加到最後面
2. empty --> 判斷queue是否empty跟判斷stack一樣，都是判斷裡面的array.length是不是0，不過題目限定只能用標準的stack方法，因此我們只能用stack.size() == 0來判斷。
3. pop --> 這比較麻煩，因爲只能用stack.pop()來實做，這邊用一個keep stack來保存stack裡面拿出來的東西，在拿出stack最後一筆資料時再倒回去。
4. peek --> 跟pop類似，直接看下面程式碼差異的部分就可以

解題

```
/**
 * @constructor
 */
var Queue = function() {
    this.stack = [];
};


// 自己在 array裡面加上stack的size方法
Array.prototype.size = function () {
  return this.length;
};


/**
 * @param {number} x
 * @returns {void}
 */
Queue.prototype.push = function(x) {
    this.stack.push(x);
};

/**
 * queue的pop()是拿出最底下一筆，stack的pop()是拿出最上面一筆
 * 因此stack不斷的pop()值到最後一筆才是我們要的資料
 * 使用另外一個stack來保存之前拿出的資料，取出之後將keep裡面的資料倒回我們的que
 * @returns {void}
 */
Queue.prototype.pop = function() {

    var keep = [];      // 儲存從stack拿出來的資料
    var element ;       // 要回傳的資料

    // 從stack裡面不斷的倒資料到keep，直到最後一筆
    while(this.stack.size() > 1){
        keep.push(this.stack.pop());
    }

    // 最後一筆是我們要回傳的資料，而且他不會回到stack中了
    var element = this.stack.pop();

    // 把keep的資料倒回stack
    while(keep.size() > 0){
        this.stack.push(keep.pop());
    }

    return element;
};

/**
 * @returns {number}
```

```
 */
Queue.prototype.peek = function() {
    var keep = [];
    var element ;

    while(this.stack.size() > 1){
        keep.push(this.stack.pop());
    }

    var element = this.stack.pop();

    // 跟pop不同的是，最後一筆要回到stack中
    keep.push(element);

    while(keep.size() > 0){
        this.stack.push(keep.pop());
    }
    return element;
}

/**
 * @returns {boolean}
 */
Queue.prototype.empty = function() {
    return this.stack.size() == 0;
};
```

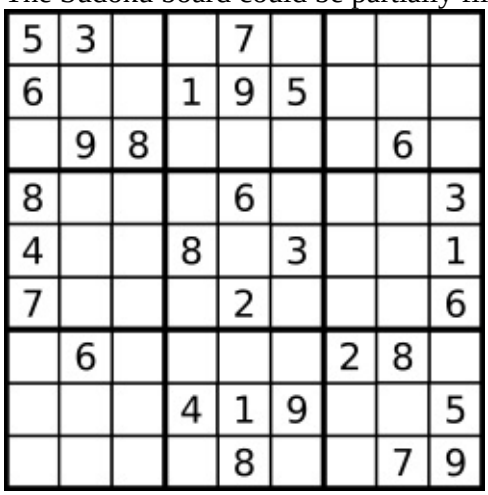
LeetCode 36. Valid Sudoku

LeetCode 36. Valid Sudoku

題目

Determine if a Sudoku is valid, according to: Sudoku Puzzles - The Rules.

The Sudoku board could be partially filled, where empty cells are filled with the character '.'.



翻譯

判斷是不是一個合法的數獨，規則請參考 Sudoku Puzzles - The Rules.

不用填滿數獨，只要判斷是否合法就可以，空的格子在這邊用'.'來代替。

思路

1. 數獨是否合法，每一個格子需要判斷的有，同row的數字不能重複，同column的數字不能重複，同一個九宮格box內不能重複三種情況要考慮
2. 將每個row，column，box都變成一行陣列，再寫一個判斷陣列裡面是否有重複值的validRepeat function
3. 讀一個數獨board使用雙層的for loop，每次讀一個row，直接把row丟進validRepeat裡面判斷是否有重複的值
4. board[0][0~8] <--第一個row內的值 ， board[0~8][0] <--第一個column內的值
5. 判斷[i,j]第幾個九宮格用 parseInt(i/3)x3+j/3，將九宮格的值用陣列儲存，最後再一個一個判斷是否有重複的值

解題

```
    ``` /**
        ◦ @param {character[][]} board
        ◦ @return {boolean} */ var isValidSudoku = function(board) { // 判斷九宮格內是否有重複數字用
          var boxs = [[],[],[],[],[],[],[],[],[]];

for(var i = 0 ; i < 9 ; i++){
    // 判斷column是否有重複數字用
    var cRow = [];

    // 直接將row丟進去判斷是否有重複
    if(!validRepeat(board[i])){
        return false;
    }

    for(var j = 0 ; j < 9 ; j++){
        cRow.push(board[j][i]);   // board[j = 0~8][i = 0] => 等於把第

        // 根據i,j判斷目前位子是屬於哪個九宮格， boxId = 3*(i/3取整數) + (j
        var boxId = 3*parseInt(i/3) +parseInt(j/3);
        boxs[boxId].push(board[i][j]);
    }

    //console.log(cRow)

    if(!validRepeat(cRow)){
        return false;
    }
}
//console.log(boxs)

// 判斷九宮格內是否有重複的值
for(var k = 0 ; k < 9 ; k++){
    if(!validRepeat(boxs[k])){
        return false;
    }
}

return true;

// 使用一個
function validRepeat(array){
    var map = [];
    for(var i = 0; i < 9 ; i++){
        if(array[i] == ".") continue;
        if(map.indexOf(array[i]) == -1){
            map.push(array[i]);
        } else {
            return false;
        }
    }
    return true;
```
```

```
}
};
```

LeetCode 299. Bulls and Cows

299. Bulls and Cows

題目

You are playing the following Bulls and Cows game with your friend:
You write down a number and ask your friend to guess what the number is. Each time your friend makes a guess, you provide a hint that indicates how many digits in said guess match your secret number exactly in both digit and position (called "bulls") and how many digits match the secret number but locate in the wrong position (called "cows"). Your friend will use successive guesses and hints to eventually derive the secret number.

For example:

```
Secret number:  "1807"
Friend's guess: "7810"
```

Hint: 1 bull and 3 cows. (The bull is 8, the cows are 0, 1 and 7.) Write a function to return a hint according to the secret number and friend's guess, use A to indicate the bulls and B to indicate the cows. In the above example, your function should return "1A3B".

Please note that both secret number and friend's guess may contain duplicate digits, for example:

```
Secret number:  "1123"
Friend's guess: "0111"
```

In this case, the 1st 1 in friend's guess is a bull, the 2nd or 3rd 1 is a cow, and your function should return "1A1B". You may assume that the secret number and your friend's guess only contain digits, and their lengths are always equal.

翻譯

這題目敘述有點長，我記得以前這是以前堂哥跟我玩的(猜數字遊戲)[http://codepen.io/skyyen999/full/VedwqQ/]，上面那一大串英文的意思是，這邊有一串隱藏的號碼(secret number)，然後你朋友會猜一串號碼， 如果號碼數字與位置都對了，給一個bull，數字對但位置不對，給一個cow。

範例:

```
Secret number:  "1807"
Friend's guess: "7810"
```

上面範例可以發現8得到一個bull，剩下1,0,7得到一個cow，所以得到1A3B

```
Secret number:  "1123"
Friend's guess: "0111"
```

第二個範例，第二個1得到一個bull，Friend's guess 第三個1得到一個cow(比對secret number的第一個1)，因此得到1A1B

思路

1. 先判斷有幾個bull(位置數字都一樣)
2. 出現bull將secret number跟friend's guess這個位子上的數字移除
3. 剩下的字串再來判斷有幾個cow

解題

```
/**
 * @param {string} secret
 * @param {string} guess
 * @return {string}
 */
var getHint = function(secret, guess) {
    var bull = 0;
    var cow  = 0;

    // 儲存secret[n],guess[n]數字不同的元素
    var skeep = [];
    var gkeep = [];

    //先判斷位置數字都一樣，剩下的用sK, gK來儲存
    for(var i in guess){
        // 位置數字都一樣，bull++
        if(secret[i]  == guess[i]){
            bull++;
        } else {
            skeep.push(secret[i]);
            gkeep.push(guess[i]);
        }
    }

    // 因爲bull已經處理過，這邊只要gkeep內的元素出現在skeep內，代表就是一個cow
    for(var j in gkeep){
        var findIndex = skeep.indexOf(gkeep[j]);
        if(findIndex != -1){
            cow++;
            skeep[findIndex] = null;
        }
    }

    return bull+ "A" + cow + "B"
};
```

LeetCode 225. Implement Stack using Queues

225. Implement Stack using Queues

題目

Implement the following operations of a stack using queues.

push(x) -- Push element x onto stack. pop() -- Removes the element on top of the stack. top() -- Get the top element. empty() -- Return whether the stack is empty.

Notes:

You must use only standard operations of a queue -- which means only push to back, peek/pop from front, size, and is empty operations are valid. Depending on your language, queue may not be supported natively. You may simulate a queue by using a list or deque (double-ended queue), as long as you use only standard operations of a queue. You may assume that all operations are valid (for example, no pop or top operations will be called on an empty stack).

翻譯

使用queue來實做 stack。

- push(x) -- 將一個元素x放入queue最後面
- pop() -- 移除stack最上面的元素
- peek() -- 取得stack最上面的元素(不刪除)
- empty() -- 檢查stack是否是空的

注意：
你只能用標準的queue方法操作，這表示只有push到queue後端，peek/pop queueu前端的元素，size以及sEmpty這些方法是可以用的。

有些語言可能沒有支援queue，你可以使用類似的結構，例如list或deque, 只要你的操作符合上述的標準queue方法。可以假設全部的操作都是合法的(例如說當stack是empty的時候，不會執行pop這樣的操作)。

思路

跟LeetCode 232. Implement Queue using Stacks幾乎一樣，直接看code就可以。

解題

```
/**
 * 使用一個自製的簡易queue來實作stack
 * @constructor
 */
var Stack = function() {
    this.queue = new Queue();
};


/**
 * queue跟stack都是把x加到最後面
 * @param {number} x
 * @returns {void}
 */
Stack.prototype.push = function(x) {
    this.queue.push(x);
};

/**
 * @returns {void}
 */
Stack.prototype.top = function() {
     var element;
     var keep  = new Queue();
     while(!this.queue.isEmpty()){
        element = this.queue.pop();
        keep.push(element);
     }

     while(!keep.isEmpty()){
        this.queue.push(keep.pop());
     }
    return element;
};

/**
 * @returns {number}
 */
Stack.prototype.pop = function() {
     var element;
     var keep  = new Queue();
     while(!this.queue.isEmpty()){
        element = this.queue.pop();
        if(!this.queue.isEmpty()){
            keep.push(element);
        }
     }

     while(!keep.isEmpty()){
        this.queue.push(keep.pop());
     }
    return element;
};
```

```
/**
 * 跟判斷queue是否爲空一樣
 * @returns {boolean}
 */
Stack.prototype.empty = function() {
    return this.queue.isEmpty();
};


/**
 * 用array實做一個簡單的Queue
 */
var Queue = function(){
    this.list = [];
}

/**
 *  push跟array的push一樣，加到最後
 */
Queue.prototype.push = function(x){
    this.list.push(x);
}

/**
 *  peek回傳第一筆資料但是不刪除
 */
Queue.prototype.peek = function(){
    return this.list[0];
}
/**
 *  pop拿出第一筆資料
 */
Queue.prototype.pop = function(){
    return this.list.shift();
}
/**
 *  判斷list裡面是不是沒東西
 */
Queue.prototype.isEmpty = function(){
    return this.list.length == 0;
}
```

LeetCode 190. Reverse Bits

LeetCode 190. Reverse Bits

題目

Reverse bits of a given 32 bits unsigned integer.

For example, given input 43261596 (represented in binary as 00000010100101000001111010011100), return 964176192 (represented in binary as 00111001011110000010100101000000).

Follow up: If this function is called many times, how would you optimize it?

翻譯

給一個32 bits的int整數，反轉整數的bits。

範例： 整數43261596 轉換成bits = 00000010100101000001111010011100，將bit反轉00111001011110000010100101000000再轉成整數964176192回傳

進階：
如果這個function會被呼叫很多次，要怎麼做最佳化?

思路

用除法將十進位數字轉換成二進位數字，也就是bits，將轉換後的二進位數字反轉，再轉回整數。

解題

```
/**
 * @param {number} n - a positive integer
 * @return {number} - a positive integer
 */
var reverseBits = function(n) {
    if(n == 0) return 0;
    // 用一個list將n轉爲2進位的bits array
    var list = [];

    for(var i = 0 ; i < 32 ; i++){
        if(n > 0){
```

```
            // 從低位數開始轉換爲2進位，放進list時就已經完成反轉的動作
            // ex. 6轉換爲2進位爲110 ( 6 => 6%2 = 0 ， 3%2 = 1 , 1%2 =1
            list.push(parseInt(n%2));
            n = parseInt(n/2);
        } else {
            list.push(0);
        }
    }

    // 將bits array轉換成整數回傳
    return parseInt(list.join(""),2);
};
```

LeetCode 303. Range Sum Query - Immutable

303. Range Sum Query - Immutable

題目

Given an integer array nums, find the sum of the elements between indices i and j (i ≤ j), inclusive.

```
Example:
Given nums = [-2, 0, 3, -5, 2, -1]
```

sumRange(0, 2) -> 1 sumRange(2, 5) -> -1 sumRange(0, 5) -> -3 </pre> Note: You may assume that the array does not change. There are many calls to sumRange function.

翻譯

給一個int陣列，寫一個sumRange方法找尋陣列中元素i~j的總和。

範例： nums = [-2, 0, 3, -5, 2, -1]
sumRange(0, 2) -> 1 sumRange(2, 5) -> -1 sumRange(0, 5) -> -3

注意：你不能改變傳入陣列的值，然後sumRange會被忽叫很多次。

思路

1. 一開始的想法是sumRange被忽叫時就跑迴圈把i到j之間全部的值加起來，這很簡單，不過果然超時(TLE)了
2. 換個想法，sumRange會被忽叫很多次，因此在裡面寫迴圈不是一個好辦法
3. 這邊先用另外一個array來儲存a[0]~a[i]的加總，之後要取加總只要取array[j]-array[i-1]就可以
4. 以 [-2, 0, 3, -5, 2, -1] 爲例，array = [-2,-2,1,-4,2,1]
5. sumRange(0, 2) --> i = 0，因此只要直接取array[j] = 2
6. sumRange(2, 5) --> 取array[5] = -1，array[1] = 0， sum = array[j]-array[i-1] = -1

解題

```
/**
 * @constructor
 * @param {number[]} nums
 */
var NumArray = function(nums) {
    var sum = 0;
```

```
    this.array = [];
    // 先將加總存到一個array
    for(var i in nums){
        sum += nums[i];
        this.array.push(sum);
    }
};

/**
 * @param {number} i
 * @param {number} j
 * @return {number}
 */
NumArray.prototype.sumRange = function(i, j) {
    if(i == 0){
        return this.array[j];
    }
    //取出的時候，只要取array[0]到array[j]的加總sumJ 減去 array[0]到array[
    return this.array[j] - this.array[i-1];
};


/**
 * Your NumArray object will be instantiated and called as such:
 * var numArray = new NumArray(nums);
 * numArray.sumRange(0, 1);
 * numArray.sumRange(0, 2);
 */
```

LeetCode 155. Min Stack

LeetCode 155. Min Stack

題目

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

push(x) -- Push element x onto stack. pop() -- Removes the element on top of the stack. top() -- Get the top element. getMin() -- Retrieve the minimum element in the stack.

```
Example:
MinStack minStack = new MinStack(); // stack = [];
minStack.push(-2);                  // stack = [-2];
minStack.push(0);                   // stack = [-2,0];
minStack.push(-3);                  // stack = [-2,0,-3];
minStack.getMin();   --> Returns -3. // stack = [-2,0,-3]
minStack.pop();                      // stack = [-2,0]，移除-3
minStack.top();      --> Returns 0.  // stack = [-2,0]
minStack.getMin();   --> Returns -2. // stack = [-2,0]
```

翻譯

設計一個支援 push, pop, top 以及在常數時間找出最小值的方法getMin, 也就是說getMin這個方法的時間複雜度爲O(1)。

push(x) -- 將元素x加入stack上面 pop() -- 移除stack最上面的元素 top() -- 取得stack最上面的元素(不移除) getMin() -- 取得stack裡面最小的元素(不移除)

思路

這題沒有提供javascript，改用java解，一開始打算無視時間複雜度, getMin的時候就跑迴圈來找出最小值，不過這樣超過時間沒辦法通過測試，這邊寫法請看下面失敗的寫法

上網搜尋後發現這個解法簡單易懂，以下分享學習：

1. 首先使用一個變數min來儲存stack中最小的值，這樣getMin()就符合O(1)，速度會快很多
2. 因此就要改成在push跟pop的地方動手腳，push的時候，如果加入的x小於等於min，除了讓min=x，我們還先把目前的min加到stack中
3. 以下用[2,3]這個堆疊作範例，目前min = 2， stack.push(1)，1小於2，所以將2先加入stack再把1加入，stack=[2,3,2,1]
4. 如此一來getMin()的時候會直接回傳1，對top()這個方法也沒任何的影響，但是pop()的時

候就會有問題了

5. stack.pop()移除最上面的元素，stack=[2,3,2]，跟正確的stack=[2,3]明顯是不一樣的，而且min也沒更新爲2
6. 這時候之前多放入的第二小的數[2]，就發揮作用了，讓min = 2，然後再把這個2移除，stack = [2,3]，min也變成了2，重此之後世界又恢復了和平

解題

```
public class MinStack {
    Stack<Integer> stack;

    // 使用一個min來紀錄stack中最小的值，注意這邊的min不能用Integer
    // Integer == Integer 的比對方法只保證再-128~127之間會有正確的結果
    int min = Integer.MAX_VALUE;

    /** initialize your data structure here. */
    public MinStack() {
        this.stack = new Stack<>();
    }

    public void push(int x) {
        //如果x <= min，先將目前的min放入stack後面，再放入x
        if(x <= min){
            this.stack.push(min);
            this.min = x;
        }
        this.stack.push(x);
    }

    public void pop() {
        // 如果取出的值剛好就是min，因爲之前我們把第二小的數放在min前面，所以現
        if(stack.peek() == this.min){
           //因爲之前x<=min的時候也壓了第二小的數進來，所以他也要被pop
           this.stack.pop();
           min =  this.stack.pop();
        } else {
           this.stack.pop();
        }

    }

    public int top() {
        return this.stack.peek();
    }

    public int getMin() {
        return min;
    }
}
```

超出時間的寫法

```
public class MinStack {
    Stack<Integer> stack;

    public MinStack() {
        this.stack = new Stack<>();
    }

    public void push(int x) {
        this.stack.push(x);
    }

    public void pop() {
        if(!stack.empty()){
           this.stack.pop();
        }
    }

    public Integer top() {
        if(!stack.empty()){
           return this.stack.peek();
        } else {
           return null;
        }
    }

    //跑迴圈找出最小值
    public int getMin() {
        int min = Integer.MAX_VALUE;
        Stack<Integer> keep = new Stack<>();

        while(!stack.empty()){
            int v = stack.pop();
            min = v < min ? v : min;
            keep.push(v);
        }

        while(!keep.empty()){
            stack.push(keep.pop());
        }
        return min;
    }
}
```