

Leetcode note

1. Backtrack problem 演算法整理
 2. 動態規劃(DP)講解：<https://www.jianshu.com/p/a7741619dd58>
 3. 未解題目：
 - 943, 996, 37, 212(used tree), 126(disgust), 675, 131(DP), 93(DP), 282(Hard), 968, 979, 99(Morris Traversal螺旋二叉樹),
 - Binary Search: { 981(BS), 378(Binary Search Solution), 668(same as 378), 778, 174, 875, 719, 786, 4 }
 - \$\$\$\$\$\$\$\$\$\$\$\$\$: { Array:(244, 245,)}
-

Trick

1. two dimension array to one dimension: $M \times N \rightarrow A[i/N][i \% N] = \text{number } i \text{ elem in array}$
2. unordered_map 是 HashMap，插入和查詢的時間複雜度都是 $O(1)$ ，但是裡面的映射對兒是無序的。
3. map 是 TreeMap，插入和查詢的時間複雜度都是 $O(\lg n)$ ，但裡面的映射對兒可以按照 key 值排序，還可以自定義排序方法。同理 set, unordered_set
4. 無條件進位 $k = (k+1)/2$ 例題23
5. if ($k++ > 0$) -> 先判斷 k 是否大於零，再 $++$; if ($++k > 0$) -> 先 $++$ 再判斷是否大於零

複習

1. priority_queue 的宣告，如何客製化比較法？
2. string to int, int to string?
3. how to use substr?

4. how to initialize 2-D vector

17. Letter Combinations of a Phone Number

Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below. Note that 1

does not map to any letters.



Example:

Input: "23"

Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

思路

input不知道會有多少數值，因此不能使用迴圈解 -> 遞迴解
字串走訪使用『範圍for』，char in string 搭配遞迴來窮舉所有可能解(dfs)

Code

```
vector<string> letterCombinations(string digits) {
    if (digits.empty()) return {};

    unordered_map<char, string> num_to_char {{'2', "abc"}, {'3', "def"}, {'4',
    vector<string> result;
    string combination(digits.length(), ' ');
    letterCombinations(digits, combination, 0, num_to_char, result);
    return result;
}

void letterCombinations(string digits, string& combination, int idx, unordered_map<char, string> num_to_char) {
    if (idx == digits.length()){
        result.push_back(combination);
        return;
    }

    for (char c : num_to_char[digits[idx]]){
        combination[idx] = c;
        letterCombinations(digits, combination, idx + 1, num_to_char, result);
    }
}
```

```
}
```



39. Combination Sum

Given a set of candidate numbers (candidates) (without duplicates) and a target number (target), find all unique combinations in candidates where the candidate numbers sums to target.

The same repeated number may be chosen from candidates unlimited number of times.

Note:

All numbers (including target) will be positive integers.

The solution set must not contain duplicate combinations.

Example 1:

Input: candidates = [2,3,6,7], target = 7,

A solution set is:

```
[  
[7],  
[2,2,3]  
]
```

Example 2:

Input: candidates = [2,3,5], target = 8,

A solution set is:

```
[  
[2,2,2,2],  
[2,3,3],  
[3,5]  
]
```

思路

排列問題走dfs，『狀態需紀錄』(只能向後看)

組合問題需考慮重複數字不同順序(也須向前看)

因為數字可重複，在擴張時要將自己也考慮進可能值中

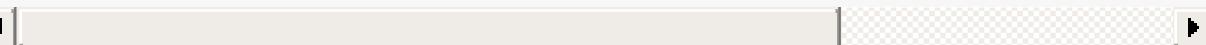
在dfs迴圈當中，可用預處理直接結束該輪(不用push再pop)

與#17不同點，#17是直接取代該值(push&pop一起做)，而此題有push因此在走到底後要pop

Code

```
class Solution {
public:
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        vector<vector<int>> answer;
        vector<int> comb;
        sort(candidates.begin(), candidates.end());
        dfs(candidates, target, answer, comb, 0);

        return answer;
    }
private:
    void dfs(vector<int>& candidates, int target, vector<vector<int>>& answer, vector<int> comb) {
        if (target == 0){
            answer.push_back(comb);
            return;
        }
        for (int i = 0; i < candidates.size(); ++i){
            if (target < candidates[i]) break; // 在push前進行預篩選
            comb.push_back(candidates[i]);
            /*for (vector<int>::const_iterator it = comb.begin(); it != comb.end();
                cout << *it << ' ';
            }
            cout << endl;*/
            dfs(candidates, target - candidates[i], answer, comb, i); // 用i來控制『只
            comb.pop_back();
        }
    }
};
```



```
class Solution:
    def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:
        def dfs(targ, ans, index):
            if targ < 0:
                return
            if targ == 0:
                res.append(ans)
                return

            for i in range(index, len(candidates)):
                dfs(targ - candidates[i], ans + [candidates[i]], i)

        res = []
```



```
dfs(target, [], 0)  
    return res
```

40. Combination Sum II

Given a collection of candidate numbers (candidates) and a target number (target), find all unique combinations in candidates where the candidate numbers sums to target.

Each number in candidates may only be used once in the combination.

Note:

All numbers (including target) will be positive integers.

The solution set must not contain duplicate combinations.

Example 1:

Input: candidates = [10,1,2,7,6,1,5], target = 8,

A solution set is:

```
[  
[1, 7],  
[1, 2, 5],  
[2, 6],  
[1, 1, 6]  
]
```

Example 2:

Input: candidates = [2,5,2,1,2], target = 5,

A solution set is:

```
[  
[1,2,2],  
[5]  
]
```

思路

1. 每個數字只能用一次，因此次態status要+1
2. Candidate中有重複的數字，因此向後看時會出現重複的數組情形
經觀察發現，不要讓重複數組情形出現的最好方式就是『此路不通時不走此路』

在pop_back之後就可以確定此路不通or已走過此路
因此用while判斷式去過濾與當前走過的路，
而為不影響使用重複的節點的可能性，判斷式也應當放在pop_back之後

Code

```
class Solution {
public:
    vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {
        vector<vector<int>> ans;
        vector<int> comb;
        sort(candidates.begin(), candidates.end());
        dfs(candidates, target, ans, comb, 0);
        return ans;
    }
private:
    void dfs(vector<int>& candidates, int target, vector<vector<int>>& ans, vector<int> comb) {
        if (target == 0){
            ans.push_back(comb);
            return;
        }
        int layer = status;
        for (int i = status; i < candidates.size(); ++i){
            if (target < candidates[i]) break;
            comb.push_back(candidates[i]);
            dfs(candidates, target-candidates[i], ans, comb, i+1);
            comb.pop_back();
            while (i+1<candidates.size() && candidates[i]==candidates[i+1]) i++; //跳過重複的數字
        }
    }
};
```

77. Combination

Given two integers n and k, return all possible combinations of k numbers out of 1 ... n.

Example:

Input: n = 4, k = 2

Output:

```
[  
[2,4],  
[3,4],
```

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

技巧

如果要append一個list進另一個list: outer.append(inner[:])

不加的話會是以位址的方式append

那麼後續再對於這個inner的操作pop(), 都會影響原本的outer_list

思路

排列組合問題老樣子走dfs

且只能向後看，因此次態的status要+1

因為沒有重複數字，向後看時不必討論是否出現重複解

Code

```
class Solution:  
    def combine(self, n: int, k: int):  
        if n <= 0 or k <= 0:  
            return []  
        def find_comb(idx, comb):  
            if len(comb) == k:  
                ret.append(comb[:])  
                return  
  
            for i in range(idx, n + 1):  
                comb.append(i)  
                find_comb(i + 1, comb)  
                comb.pop()  
  
        ret = []  
        for i in range(1, n + 1):  
            find_comb(i+1, [i])  
  
        return ret
```

```
class Solution {  
public:
```

```
vector<vector<int>> combine(int n, int k) {
    vector<int> comb;
    vector<vector<int>> ans;
    dfs(comb, ans, n, k, 1);
    return ans;
}
private:
    void dfs(vector<int>& comb, vector<vector<int>>& ans, int n, int k, int status)
    if (comb.size() == k){
        ans.push_back(comb);
        return;
    }
    for ( int i=status; i <= n; ++i){
        comb.push_back(i);
        dfs(comb, ans, n, k, i+1);
        comb.pop_back();
    }
}
};
```

78. Subsets

Given a set of distinct integers, nums, return all possible subsets (the power set).

Note: The solution set must not contain duplicate subsets.

Example:

Input: nums = [1,2,3]

Output:

```
[  
[3],  
[1],  
[2],  
[1,2,3],  
[1,3],  
[2,3],  
[1,2],  
[]  
]
```

思路

不必設終止條件，純粹dfs走訪就能解

Code

```
....  
ret = [[1,2,3],[1,2],[1,3],[1]]  
  
cand, idx = [1], 1  
    cand, idx = [1,2], 2  
        cand, idx = [1,2,3], 3  
            cand, idx = [1,3], 3  
  
....  
  
class Solution:  
    def subsets(self, nums: List[int]) -> List[List[int]]:  
        ret = []  
        ret.append([])  
        def recur(cand, idx):  
            if idx == len(nums):  
                return cand  
            for i in range(idx, len(nums)):  
                ret.append(recur(cand+[nums[i]], i+1))  
            return cand  
  
        for i in range(len(nums)):  
            ret.append(recur([nums[i]], i+1))  
        return ret
```

```
class Solution {  
public:  
    vector<vector<int>> subsets(vector<int>& nums) {  
        vector<vector<int>> ans;  
        vector<int> comb;  
        //ans.push_back(comb);  
        dfs(nums, ans, comb, 0, 1);  
        return ans;  
    }  
private:  
    void dfs(vector<int>& nums, vector<vector<int>>& ans, vector<int>& comb, int st  
    //if (comb.size() == level){  
        ans.push_back(comb);  
        /*for(vector<int>::iterator it=comb.begin(); it!=comb.end(); ++it){  
            cout << *it << " ";  
        }  
        cout << endl;*/  
        //      return;  
    //}  
    //for( int j = level; j <= nums.size(); ++j){
```

```

        for (int i = status; i < nums.size(); ++i){
            comb.push_back(nums[i]);
            dfs(nums, ans, comb, i+1, 0);
            comb.pop_back();
        }
    // }
};


```

90. Subsets II

Given a collection of integers that might contain duplicates, nums, return all possible subsets (the power set).

Note: The solution set must not contain duplicate subsets.

Example:

Input: [1,2,2]

Output:

```
[
[2],
[1],
[1,2,2],
[2,2],
[1,2],
[]]
```

思路

求子集合還是單存dfs走訪且不用設限制，
處理重複元素的處理，在pop_back後檢查下個元素重複的可能性

Code

```

class Solution:
    def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:
        ret = []

```

```

        self.dfs(sorted(nums), ret, [], 0)
        return ret

def dfs(self, nums, ret, cand, idx):
    ret.append(cand)
    for i in range(idx, len(nums)):
        if i > idx and nums[i] == nums[i-1]:
            continue
        self.dfs(nums, ret, cand+[nums[i]], i+1)

```

```

class Solution {
public:
    vector<vector<int>> subsetsWithDup(vector<int>& nums) {
        vector<vector<int>> ans;
        vector<int> comb;
        sort(nums.begin(), nums.end());
        dfs(nums, ans, comb, 0);
        return ans;
    }
private:
    void dfs(vector<int>& nums, vector<vector<int>>& ans, vector<int>& comb, int st
        ans.push_back(comb);
        for (int i=status; i < nums.size(); i++){
            comb.push_back(nums[i]);
            dfs(nums, ans, comb, i+1);
            comb.pop_back();
            while(i+1 < nums.size() && nums[i] == nums[i+1]) i++;
        }
    }
};

```

216. Combination Sum III

Find all possible combinations of k numbers that add up to a number n, given that only numbers from 1 to 9 can be used and each combination should be a unique set of numbers.

Note:

All numbers will be positive integers.

The solution set must not contain duplicate combinations.

Example 1:

Input: k = 3, n = 7

Output: [[1,2,4]]

Example 2:

Input: k = 3, n = 9

Output: [[1,2,6], [1,3,5], [2,3,4]]

思路

Code

```
class Solution {
public:
    vector<vector<int>> combinationSum3(int k, int n) {
        vector<vector<int>> ans;
        vector<int> comb;
        dfs(ans, comb, k, n, 1);
        return ans;
    }
private:
    void dfs(vector<vector<int>& ans, vector<int>& comb, int k, int n, int status) {
        if (comb.size() == k && n == 0){
            ans.push_back(comb);
            return;
        }
        for (int i=status; i<10; i++){
            //cout << "n: " << n << endl;
            if (n - i < 0) break;
            comb.push_back(i);
            /*for(vector<int>::iterator it=comb.begin(); it!=comb.end(); it++){
                cout << *it << " ";
            }*/
            cout << endl;
            dfs(ans, comb, k, n-i, i+1);
            comb.pop_back();
        }
    }
};
```

46. Permutations

Given a collection of distinct integers, return all possible permutations.

Example:

Input: [1,2,3]

Output:

```
[  
[1,2,3],  
[1,3,2],  
[2,1,3],  
[2,3,1],  
[3,1,2],  
[3,2,1]  
]
```

思路

permutation用dfs方式去解，
在set裡搜尋時要從頭開始找起，
因此需要maintain一個table記錄著元素用過與否

Code

```
class Solution {  
public:  
    vector<vector<int>> permute(vector<int>& nums) {  
        vector<vector<int>> ans;  
        vector<int> comb;  
        vector<bool> utable;  
        for(int i=0; i<nums.size(); i++){  
            utable.push_back(false);  
        }  
        dfs(nums, ans, comb, utable);  
        return ans;  
    }  
private:  
    void dfs(vector<int>& nums, vector<vector<int>>& ans, vector<int>& comb, vector<bool> &utable){  
        if(comb.size() == nums.size()){  
            ans.push_back(comb);  
            return;  
        }  
        for(int i = 0; i<nums.size(); i++){  
            if(utable[i]) continue;  
            comb.push_back(nums[i]);  
            utable[i] = true;  
            /*for(vector<int>::iterator it=comb.begin(); it!=comb.end(); it++){  
                cout<< *it << " ";  
            }  
            cout << endl;*/  
            dfs(nums, ans, comb, utable);  
            utable[i] = false;  
            comb.pop_back();  
        }  
    }  
}
```

```
        }
    }
};
```

784. Letter Case Permutation

Given a string S, we can transform every letter individually to be lowercase or uppercase to create another string. Return a list of all possible strings we could create.

Examples:

Input: S = "a1b2"

Output: ["a1b2", "a1B2", "A1b2", "A1B2"]

Input: S = "3z4"

Output: ["3z4", "3Z4"]

Input: S = "12345"

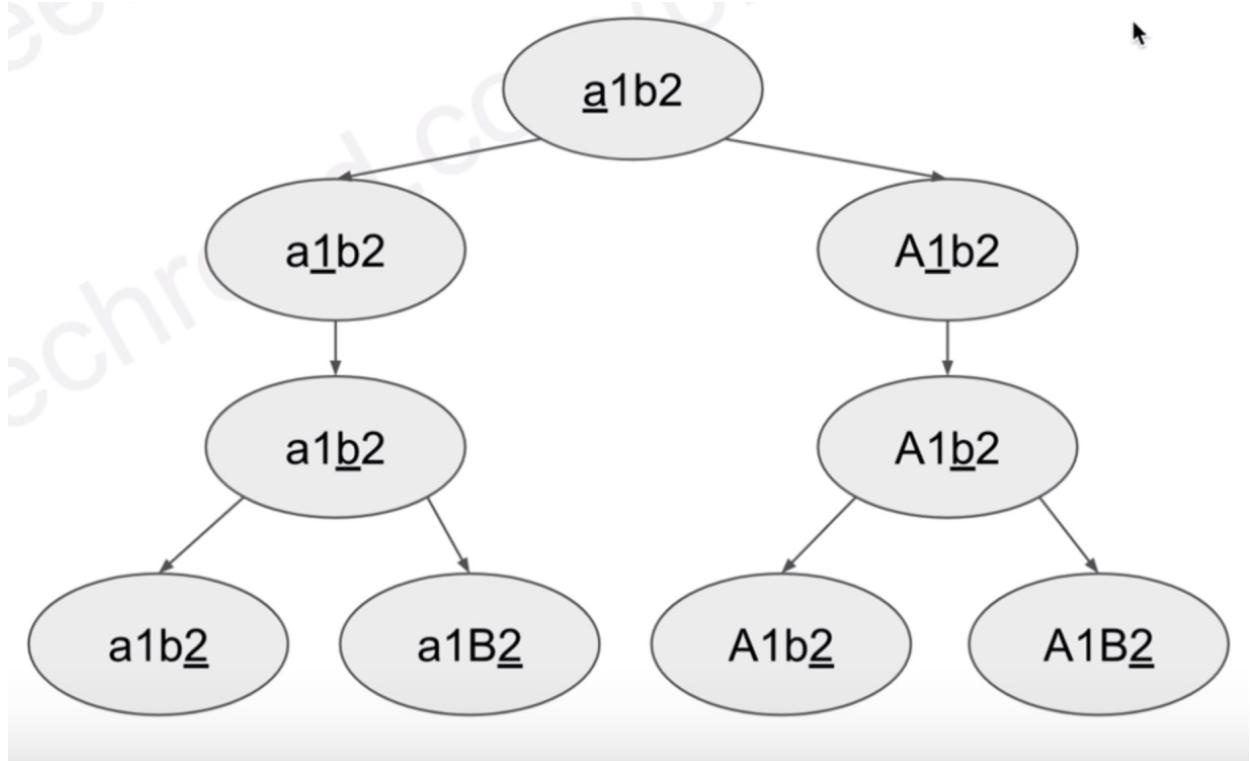
Output: ["12345"]

Note:

S will be a string with length between 1 and 12.

S will consist only of letters or digits.

思路



用dfs走訪，

注意要將改過的字串修正回來，

不然這棵樹將只會走一邊(另一端子樹沒有建立)

Code

```

class Solution {
public:
    vector<string> letterCasePermutation(string S) {
        vector<string> ans;
        dfs(ans, S, 0);
        return ans;
    }
private:
    void dfs(vector<string>& ans, string S, int status){
        ans.push_back(S);
        for (int i=status; i < S.length(); i++){
            if (isalpha(S[i])){
                if (isupper(S[i])) S[i] = tolower(S[i]);
                else S[i] = toupper(S[i]);
                dfs(ans, S, i+1);

                // 要考慮到另一邊的子樹
                if (isupper(S[i])) S[i] = tolower(S[i]);
                else S[i] = toupper(S[i]);
            }
        }
    }
};
  
```

22. Generate Parentheses

Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

For example, given n = 3, a solution set is:

```
[  
"((0))",  
"(00)",  
"(0)0",  
"0(0)",  
"000"  
]
```

思路

Backtrack 的精髓就是現在這個解在backtrack之後可以發展出其他可能的解

Recursion 考慮的三要素：Choice, Constrain, Goal

Choice: place (或)

Constrain: 左括不能超過n個, 在組合中左括一定要大於等於右括

Goal: 左右括都用盡

Note:

1. 在遞迴中狀態都必須是“可回溯的”
e.g. 不會有 `dfs(str += "(")`, 因為當`+=`的時候, 已經破壞這個狀態了
2. 只有在有一個集合set必須去走訪時, 才需要用到for(純粹的遞迴式不一定要存在for的)

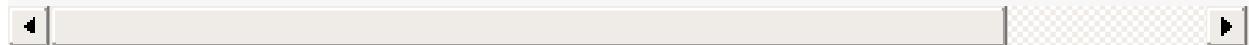
Code

```
class Solution {  
public:  
    vector<string> generateParenthesis(int n) {  
        vector<string> ans;  
        //string comb = "";  
        //int left, right = n;  
        dfs(ans, "", n, n);  
        return ans;  
    }  
}
```

```

private:
    void dfs(vector<string>& ans, string comb, int left,int right){
        /*
            We can add this answer and then backtrack so the previous call can exhaust
            more possibilities and express more answers...and then return to its call
            etc. etc.
            Yeah...this is what backtracking is all about.
        */
        if (left == 0 && right == 0){
            ans.push_back(comb);
            return;
        }
        if (left > 0)   dfs(ans, comb + "(", left-1, right); // 只要左有剩就可以放左
        if (right > left)  dfs(ans, comb + ")", left, right-1); // 一定要有一個open
    }
};

```



301. Remove Invalid Parentheses

Remove the minimum number of invalid parentheses in order to make the input string valid.
Return all possible results.

Note: The input string may contain letters other than the parentheses (and).

Example 1:

Input: "()()()

Output: ["()", "(())"]

Example 2:

Input: "(a)())()

Output: ["(a)()", "(a())()"]

Example 3:

Input: ")("

Output: [""]

思路

Code

```

class Solution {
public:
    vector<string> removeInvalidParentheses(string s) {
        int left, right = 0;
        vector<string> ans;
        getRemoveCount(s, left, right);
        dfs(ans, left, right, s, 0);
        if(ans.size() == 0) ans.push_back("");
        return ans;
    }

private:
    void getRemoveCount(string s, int& left, int& right){
        // 計算總共要刪除多少個左右括號 前綴右括號&後綴左括號
        for (char ch: s){
            if (ch == '('){
                left++;
            }
            if (ch == ')'){
                if (left == 0){ // 沒有左括號可以匹配
                    right++;
                }
                else{
                    left--;
                }
            }
        }
        cout << left << " " << right << endl;
    }
    bool isValid(string comb){
        int left, right = 0;
        for (char ch: comb){
            if (ch == '('){
                left++;
            }
            if (ch == ')'){
                if (left == 0){ // 沒有左括號可以匹配
                    right++;
                }
                else{
                    left--;
                }
            }
        }
        return (left == 0 && right == 0);
    }
    void dfs(vector<string>& ans, int left,int right, string comb, int status){
        if (left == 0 && right == 0){
            cout << comb << endl;
            if (isValid(comb)){
                ans.push_back(comb);
            }
        }
        return;
    }
}

```

```

        for (int i = status; i < comb.length(); ++i){
            if (i > 0 && comb[i] == comb[i-1])    continue;

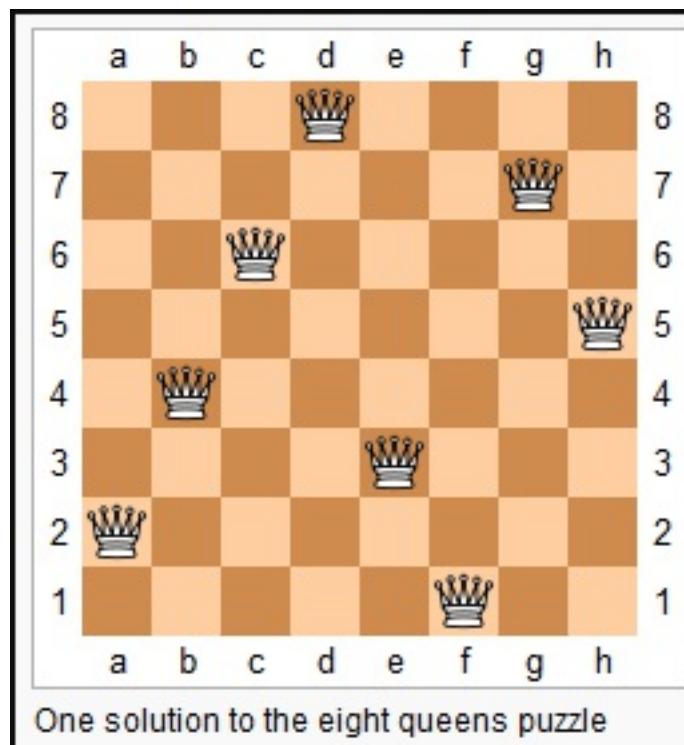
            if(comb[i] == '(' || comb[i] == ')'){
                string curr = comb;
                curr.erase(i,1);

                if (comb[i] == ')' && right > 0){
                    dfs(ans, left, right-1, curr, i);
                }
                else if (comb[i] == '(' && left > 0){
                    dfs(ans, left-1, right, curr, i);
                }
            }
        }
    };

```

51. N-Queens

The n-queens puzzle is the problem of placing n queens on an $n \times n$ chessboard such that no two queens attack each other.



Given an integer n, return all distinct solutions to the n-queens puzzle.

Each solution contains a distinct board configuration of the n-queens' placement, where 'Q'

and '.' both indicate a queen and an empty space respectively.

Example:

Input: 4

Output: [

```
[".Q..", // Solution 1  
"...Q",  
"Q...",  
"..Q."],
```

```
[..Q., // Solution 2
```

```
"Q...",  
"...Q",  
.Q.."]
```

Explanation: There exist two distinct solutions to the 4-queens puzzle as shown above.

思路

首先必須選擇以column或以role來當作遞迴的標準

1. Recursion三要素: Choice, Constrain, Goal
 - i. Choice: Place Q or .
 - ii. Constrain: 不能讓皇后衝突
 - iii. Goal: 固定邊走訪完畢
2. Previous State: 如果會更改遞迴內元素的狀態，必須在return後做恢復上一態
 - i. 不會更改的例子: [Leetcode 22](#)
3. 矩陣內135度方向元素(x,y) , $x+y = x_1+y_1$
4. 矩陣內45度方向元素(x,y) , $x-y = x_1-y_1$

Code

```
class Solution {  
public:  
    vector<vector<string>> solveNQueens(int n) {  
        vector<string> comb(n, string(n, '.'));  
        vector<vector<string>> ans;  
        backtrack(n, comb, ans, 0);  
        return ans;
```

```

    }

private:
    void backtrack(int n, vector<string> comb, vector<vector<string>>& ans, int row
        if (row == n){
            ans.push_back(comb);
            return;
        }
        for (int col=0; col<n; ++col){
            if( check(comb, row, col, n) ){
                comb[row][col] = 'Q';
                backtrack(n, comb, ans, row+1);
                comb[row][col] = '.';
            }
        }
    }

bool check(vector<string> comb, int row, int col, int n){
    // check row
    for (int i=0; i<n; ++i){
        if (i == col) continue;
        if (comb[row][i] == 'Q') return false;
    }

    // check col
    for (int i=0; i<n; ++i){
        if (i == row) continue;
        if (comb[i][col] == 'Q') return false;
    }

    // check diag
    for (int i=0; i<n; ++i){
        if (i == row) continue;

        // check diag+ 斜率為正的對角 row+col (1,2) -> (0,3), (2,1), (3,0)
        if ( (row+col-i) >= 0 && (row+col-i) < n){
            if ( comb[i][row+col-i] == 'Q' ) return false;
        }

        // check diag- 斜率為負的對角 row-col (1,2) -> (0,1), (2,3)
        // i - x = row - col => x = i-row+col
        if ( (i-row+col) >= 0 && (i-row+col) < n){
            if ( comb[i][i-row+col] == 'Q' ) return false;
        }
    }
    return true;
}

};


```

79. Word Search(Medium) | 4/1

Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

Example:

```
board =  
[  
['A','B','C','E'],  
['S','F','C','S'],  
['A','D','E','E']  
]
```

Given word = "ABCCED", return true.

Given word = "SEE", return true.

Given word = "ABCB", return false.

思路

1. 雖adjacent cell表示相鄰的字，但若整個字串都要adjacent的話，則概念就轉化成 "一筆畫" 連完字串
2. 一筆畫的概念就能夠使用遞迴的解法，這個字在這個位置存在的情況下去看上下左右是否存在下個字，遇到死路就回到上一步繼續找其他方向
3. Main function: 找起點
4. 用過的不能再用：標記為'#'，若是不歸路 要改回原來的字

Code

```
class Solution {  
public:  
    bool exist(vector<vector<char>>& board, string word) {  
        int m = board.size(), n = board[0].size();  
        // Loop-searching for the start of the seq  
        for (int i=0; i<m; i++){  
            for (int j=0; j<n; j++){  
                if ( search( board, word, 0, i, j ) ) return true;  
            }  
        }  
        return false;  
    }
```

```

    }

private:
    bool search( vector<vector<char>>& board, string word, int idx, int i, int j){
        if (idx == word.size()) return true;
        int m = board.size(), n = board[0].size();
        if ( i<0 || j<0 || i>=m || j>=n || board[i][j] != word[idx]) return false;
        char temp_char = board[i][j];
        board[i][j] = '#';
        bool res = search( board, word, idx+1, i-1, j) ||
                   search( board, word, idx+1, i+1, j) ||
                   search( board, word, idx+1, i, j-1) ||
                   search( board, word, idx+1, i, j+1);
        board[i][j] = temp_char; // search fail then backtrack
        return res;
    }

};

```

127. Word Ladder(Medium) | 4/1

Given two words (beginWord and endWord), and a dictionary's word list, find the length of shortest transformation sequence from beginWord to endWord, such that:

Only one letter can be changed at a time.

Each transformed word must exist in the word list. Note that beginWord is not a transformed word.

Note:

Return 0 if there is no such transformation sequence.

All words have the same length.

All words contain only lowercase alphabetic characters.

You may assume no duplicates in the word list.

You may assume beginWord and endWord are non-empty and are not the same.

Example 1:

Input:

```

beginWord = "hit",
endWord = "cog",
wordList = ["hot", "dot", "dog", "lot", "log", "cog"]

```

Output: 5

Explanation: As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog", return its length 5.

Example 2:

Input:

```
beginWord = "hit"  
endWord = "cog"  
wordList = ["hot", "dot", "dog", "lot", "log"]
```

Output: 0

Explanation: The endWord "cog" is not in wordList, therefore no possible transformation.

思路

Code

```
class Solution {  
public:  
    int ladderLength(string beginWord, string endWord, vector<string>& wordList) {  
        unordered_set<string> wordset(wordList.begin(), wordList.end());  
        if (!wordset.count(endWord))    return 0;  
        unordered_map<string, int> pathCnt;  
        pathCnt[beginWord] = 1;  
  
        queue<string> word_need2Cnt;  
        word_need2Cnt.push(beginWord);  
  
        while(!word_need2Cnt.empty()){  
            string word = word_need2Cnt.front(); word_need2Cnt.pop();  
            for (int i=0; i<word.length(); i++){  
                for (char c='a'; c <= 'z'; c++){  
                    string new_word = word;  
                    new_word[i] = c;  
  
                    if ( wordset.count(new_word) && new_word == endWord)    return  
                    if ( wordset.count(new_word) && !pathCnt.count(new_word)){  
                        pathCnt[new_word] = pathCnt[word] +1;  
                        word_need2Cnt.push(new_word);  
                    }  
                }  
            }  
        }  
        return 0;  
    }  
};
```

因為使用BFS，可以不需要用hashmap來存path count
BFS如同漣漪般向外擴散，同一層的所有可能字的path count皆為相同
因此只需要採用迴圈去計數即可
但須注意用過的元素必須從字典裡刪除，避免出現無限迴圈的情形

```
class Solution {
public:
    int ladderLength(string beginWord, string endWord, vector<string>& wordList) {
        unordered_set<string> wordSet(wordList.begin(), wordList.end());
        if (!wordSet.count(endWord)) return 0;
        queue<string> q{{beginWord}};
        int res = 0;
        while (!q.empty()) {
            for (int k = q.size(); k > 0; --k) {
                string word = q.front(); q.pop();
                if (word == endWord) return res + 1;
                for (int i = 0; i < word.size(); ++i) {
                    string newWord = word;
                    for (char ch = 'a'; ch <= 'z'; ++ch) {
                        newWord[i] = ch;
                        if (wordSet.count(newWord) && newWord != word) {
                            q.push(newWord);
                            wordSet.erase(newWord); // 用過的元素必須要清掉，否則會陷入無
                        }
                    }
                }
                ++res;
            }
        }
        return 0;
    }
};
```

使用雙向BFS 尋找中間點
要進行BFS前先進行判斷哪條可以進行比較少的判斷

```
class Solution {
public:
    int ladderLength(string beginWord, string endWord, vector<string>& wordList) {
        unordered_set<string> wordset(wordList.begin(), wordList.end());
        unordered_set<string> headset, tailset, *phead, *ptail;
        if (wordset.find(endWord) == wordset.end()) return 0;
        headset.insert(beginWord);
        tailset.insert(endWord);
        int ladder = 2;
```

```

        while (!headset.empty() && !tailset.empty()){ // NOTE: is "&&" rather than
            // Use the less size set to first bfs
            if (headset.size() < tailset.size()){
                phead = &headset;
                ptail = &tailset;
            }
            else{
                phead = &tailset;
                ptail = &headset;
            }
            unordered_set<string> newWord2insert2wordset;
            for (auto it = phead -> begin(); it != phead -> end(); it++){
                string word = *it;
                for (int i=0; i<word.size(); i++){
                    char tempchar = word[i];
                    for (int j=0; j<26; j++){
                        word[i] = 'a' + j;
                        if ( ptail -> find(word) != ptail -> end()){
                            return ladder;
                        }
                        if ( wordset.find(word) != wordset.end()){
                            newWord2insert2wordset.insert(word);
                            wordset.erase(word); // ERASE!
                        }
                    }
                    word[i] = tempchar;
                }
            }
            ladder++;
            phead -> swap(newWord2insert2wordset);
        }
        return 0;
    }
};

```

752. Open the Lock(Medium) | 4/7

You have a lock in front of you with 4 circular wheels. Each wheel has 10 slots: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'. The wheels can rotate freely and wrap around: for example we can turn '9' to be '0', or '0' to be '9'. Each move consists of turning one wheel one slot.

The lock initially starts at '0000', a string representing the state of the 4 wheels.

You are given a list of deadends dead ends, meaning if the lock displays any of these codes, the wheels of the lock will stop turning and you will be unable to open it.

Given a target representing the value of the wheels that will unlock the lock, return the

minimum total number of turns required to open the lock, or -1 if it is impossible.

Example 1:

Input: deadends = ["0201", "0101", "0102", "1212", "2002"], target = "0202"

Output: 6

Explanation:

A sequence of valid moves would be "0000" -> "1000" -> "1100" -> "1200" -> "1201" -> "1202" -> "0202".

Note that a sequence like "0000" -> "0001" -> "0002" -> "0102" -> "0202" would be invalid, because the wheels of the lock become stuck after the display becomes the dead end "0102".

Example 2:

Input: deadends = ["8888"], target = "0009"

Output: 1

Explanation:

We can turn the last wheel in reverse to move from "0000" -> "0009".

Example 3:

Input: deadends = ["8887", "8889", "8878", "8898", "8788", "8988", "7888", "9888"], target = "8888"

Output: -1

Explanation:

We can't reach the target without getting stuck.

Example 4:

Input: deadends = ["0000"], target = "8888"

Output: -1

Note:

The length of deadends will be in the range [1, 500].

target will not be in the list deadends.

Every string in deadends and the string target will be a string of 4 digits from the 10,000 possibilities '0000' to '9999'.

思路

0. 與#127概念類似
1. 雙向BFS解
2. Number to string
code[i] = ((temp[i] - '0') + 10 + j) % 10 + '0';

Code

```
class Solution {  
public:  
    int openLock(vector<string>& deadends, string target) {
```

```

if (deadset.find("0000") != deadset.end()) return -1;
unordered_set<string> headset, tailset, visited, *phead, *ptail;
unordered_set<string> deadset(deadends.begin(), deadends.end());
int res=0;
headset.insert("0000");
visited.insert("0000");
tailset.insert(target);

while ( !headset.empty() && !tailset.empty()){
    unordered_set<string> branch;
    if (headset.size() < tailset.size()){
        phead = &headset;
        ptail = &tailset;
    }
    else{
        phead = &tailset;
        ptail = &headset;
    }

    for (auto it = phead -> begin(); it != phead -> end(); it++){
        string code = *it;
        for (int i=0; i<4; i++){
            string temp = code;
            for (int j=-1; j<=1; j++){
                if (j == 0) continue; // ignore the visited one
                code[i] = ((temp[i] - '0') + 10 + j) % 10 + '0';

                if (ptail -> find(code) != ptail -> end() ) return res+1;
                if (deadset.find(code) == deadset.end() && visited.find(code) == visited.end())
                    branch.insert(code);
                visited.insert(code);
            }
        }
        code = temp;
    } // end of +/- 1
    res++;
    phead -> swap(branch);
}
return -1;
};


```

542. 01 Matrix (Medium) | 4/7

Given a matrix consists of 0 and 1, find the distance of the nearest 0 for each cell.

The distance between two adjacent cells is 1.

Example 1:

Input:

0 0 0

0 1 0

0 0 0

Output:

0 0 0

0 1 0

0 0 0

Example 2:

Input:

0 0 0

0 1 0

1 1 1

Output:

0 0 0

0 1 0

1 2 1

Note:

The number of elements of the given matrix will not exceed 10,000.

There are at least one 0 in the given matrix.

The cells are adjacent in only four directions: up, down, left and right.

思路

1. 比較直覺的方式：使用BFS去尋找最短路徑
2. 透過已知去拜訪未知，從0的觀點去看1
 - 用queue將已知的距離記錄起來
 - 先將0記錄起來，距離為0
 - 將拜訪過的距離放入queue中，以供後續使用
 - 並且記錄為拜訪過

Code

bfs

```
class Solution:  
    def updateMatrix(self, matrix: List[List[int]]) -> List[List[int]]:  
        r_b, c_b = len(matrix), len(matrix[0])
```

```

queue, visited = [], set()
for i in range(r_b):
    for j in range(c_b):
        if matrix[i][j] == 0:
            queue.append((i,j))
            visited.add((i,j))

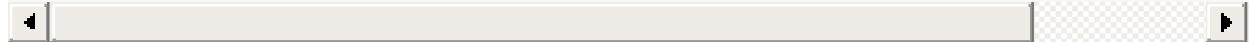
for row, col in queue:
    distance = matrix[row][col] + 1

    for dx, dy in [(1,0), (-1,0), (0,1), (0,-1)]:
        n_x, n_y = row + dx, col + dy # take a look at its neighborhood

        # not in visited means is 1 and not visit before
        if 0 <= n_x < r_b and 0 <= n_y < c_b and (n_x, n_y) not in visited:
            matrix[n_x][n_y] = distance
            queue.append((n_x, n_y)) # keep the record of already calculate
            visited.add((n_x, n_y))

return matrix

```



dfs TLE

```

class Solution:
    def updateMatrix(self, matrix: List[List[int]]) -> List[List[int]]:
        direction = [(0,1), (1,0), (0,-1), (-1,0)]
        def distance(i, j, count, min_dis):
            if matrix[i][j] == 0:
                return count

            for k in range(len(direction)):
                x, y = i+direction[k][0], j+direction[k][1]
                if 0 <= x < r and 0 <= y < c:
                    if count + 1 < min_dis:
                        min_dis = min(min_dis, distance(x, y, count+1, min_dis))

        return min_dis

        r, c = len(matrix), len(matrix[0])
        for i in range(r):
            for j in range(c):
                if matrix[i][j] == 1:
                    matrix[i][j] = distance(i,j, 0, 201)

        return matrix

```

用BFS找最短路徑

```

class Solution {
public:
    vector<vector<int>> updateMatrix(vector<vector<int>>& matrix) {
        int m = matrix.size(), n = matrix[0].size();
        int dirs[5] = {0, -1, 0, 1, 0}; // simple way to record of direction matrix
        queue<pair<int, int>> points2calc;

        for (int i=0; i<m; i++){
            for (int j=0; j<n; j++){
                if (matrix[i][j] == 0){
                    points2calc.push({i,j}); // BFS from 0
                }
                else if (matrix[i][j] == 1){
                    matrix[i][j] = 201; // num of elements given matrix not exceed
                }
            }
        }

        while (!points2calc.empty()){
            auto t = points2calc.front(); points2calc.pop();
            for (int i=0; i<4; i++){
                int x = t.first + dirs[i];
                int y = t.second + dirs[i+1];

                // (x,y) > (t.f,t.s)+1 means there is a shorter path value (t.f,t.s)+1
                if (x>=0 && y>=0 && x<m && y<n &&
                    matrix[x][y] > matrix[t.first][t.second] + 1){
                    matrix[x][y] = matrix[t.first][t.second] + 1;
                    points2calc.push({x,y});
                }
            }
        }

        return matrix;
    }
};

```

使用兩階段比較法(牛B):

```

```
class Solution {
public:
 vector<vector<int>> updateMatrix(vector<vector<int>>& matrix) {
 int m = matrix.size(), n = matrix[0].size();
 vector<vector<int>> res(m, vector<n, INT_MAX - 1>);

 // 比較左&上 從左上走到右下
 for (int i = 0; i < m; ++i) { for (int j = 0; j < n; ++j) { if (matrix[i][j] == 0) res[i][j] = 0; else { if (i > 0)
 res[i][j] = min(res[i][j], res[i - 1][j] + 1);
 if (j > 0) res[i][j] = min(res[i][j], res[i][j - 1] + 1);
 }
 }
}
```

```

```

}
}

}

// 比較右&下 從右下走到左上
// 到第二階段，若值已經為0或1則不可能找到更短的距離因此不用更新
for (int i = m - 1; i >= 0; --i) {
    for (int j = n - 1; j >= 0; --j) {
        if (res[i][j] != 0 && res[i][j] != 1) {
            if (i < m - 1) res[i][j] = min(res[i][j], res[i + 1][j] + 1);
            if (j < n - 1) res[i][j] = min(res[i][j], res[i][j + 1] + 1);
        }
    }
}
return res;
}
};


```

934. Shortest Bridge(Medium) / 4/7
 In a given 2D binary array A, there are two islands. (An island is a 4-directional
 Now, we may change 0s to 1s so as to connect the two islands together to form 1 isl
 Return the smallest number of 0s that must be flipped. (It is guaranteed that the

Example 1:

Input: [[0,1],[1,0]]

Output: 1

Example 2:

Input: [[0,1,0],[0,0,0],[0,0,1]]

Output: 2

Example 3:

Input: [[1,1,1,1,1],[1,0,0,0,1],[1,0,1,0,1],[1,0,0,0,1],[1,1,1,1,1]]

Output: 1

Note:

1 <= A.length = A[0].length <= 100

A[i][j] == 0 or A[i][j] == 1

思路

)

```
### Code
```

```
c++  
class Solution {  
public:  
int shortestBridge(vector<vector<int>>& A) {  
// identify the other island  
for (int i=0, found=0; !found && i<A.size(); i++){  
for (int j=0; !found && j<A[0].size(); j++){  
found = paint(A, i, j);  
}  
}  
}
```

```
for (int cl=2; ;cl++){  
    for (int i=0, found=0; !found && i<A.size(); i++){  
        for (int j=0; !found && j<A[0].size(); j++){  
            if (cl == A[i][j] && (expand(A, i-1, j, cl) || expand(A, i, j-1, cl)  
                || expand(A, i+1, j, cl) || expand(A, i, j+1, cl))  
            {  
                return cl-2;  
            }  
        }  
    }  
}
```

```
private:  
int paint(vector<vector<int>>& A, int i, int j){  
if (i<0 || j<0 || i>= A.size() || j>= A[0].size() || A[i][j] != 1) return 0;  
A[i][j] = 2;  
return 1 + paint(A, i-1, j) + paint(A, i, j-1) + paint(A, i+1, j) + paint(A, i, j+1);  
}
```

```
bool expand(vector<vector<int>>& A, int i, int j, int cl){  
    if (i<0 || j<0 || i>= A.size() || j>= A[0].size())  
        return false;  
    if (A[i][j] == 0)  
        A[i][j] = cl+1;  
    return A[i][j] == 1;  
}
```

```
};
```

698. Partition to K Equal Sum Subsets(Medium) / 4/8
Given an array of integers `nums` and a positive `integer k`, find whether it's possibl

Example 1:

Input: `nums = [4, 3, 2, 3, 5, 2, 1]`, `k = 4`

Output: True

Explanation: It's possible to divide it into 4 subsets (5), (1, 4), (2,3), (2,3) wi

Note:

`1 <= k <= len(nums) <= 16.`

`0 < nums[i] < 10000.`

思路

首先判斷sum是否能整除k，不能整除的話直接返回`false`。然後需要一個`visited`數組來記錄哪些數組已經被選中。

一些優化：比如先給數組按從大到小的順序排個序，然後在遞歸函數中，我們可以直接判斷，如果`curSum`大於`target`，直接回

Code

C++

```
class Solution {  
public:  
    bool canPartitionKSubsets(vector<int>& nums, int k) {  
        int sum = accumulate(nums.begin(), nums.end(), 0);  
        if (sum % k != 0) return false;  
        sort(nums.begin(), nums.end(), greater());  
        vector<bool> visited(nums.size(), false);  
        return dfs(nums, visited, sum/k, 0, 0, k);  
    }  
    bool dfs(vector<int>& nums, vector<bool>& visited, int target, int curSum, int idx, int goal){  
        if (goal == 1) return true;  
        if (curSum > target) return false;  
        if (curSum == target) return dfs(nums, visited, target, 0, 0, goal-1);  
        for (int i=idx; i<nums.size(); ++i){  
            if (visited[i]) continue;  
            visited[i] = true;  
            if (dfs(nums, visited, target, curSum + nums[i], i+1, goal) )  
                return true;  
            visited[i] = false;  
        }  
    }  
};
```

```
return false;  
}  
};
```

93. Restore IP Addresses / 4/9
Given a string containing only digits, restore it by returning all possible valid IP addresses.

Example:

Input: "25525511135"
Output: ["255.255.11.135", "255.255.111.35"]
思路
1. 只要遇到字符串的子序列或配准問題首先考慮動態規劃DP
2. 只要遇到需要求出所有可能情況首先考慮用遞歸。

這道題並非是求字符串的子序列或配准問題，更符闡第二種情況，所以我們要用遞歸來解。我們用k來表示當前還剩多少位數。

Code



C++

```
class Solution {  
public:  
vector<string> restoreIpAddresses(string s) {  
vector<string> res;  
dfs(s, 0, "", res);  
return res;  
}  
private:  
void dfs(string s, int cnt, string comb, vector<string>& res){  
if (cnt == 4){  
if (s.empty()) res.push_back(comb);  
}  
//if (cnt == 4 && s.empty()){ res.push_back(comb); }/  
if (s.size() > (4-cnt)*3) return;  
if (s.size() < (4-cnt)) return; // possibility of digit from 1~3 for (int i=1; i<4; i++){ if (s.size() < i)  
break; int val = atoi(s.substr(0, i).c_str()); (val > 255 || i != to_string(val).size()) // val>255 or val =  
01, 001, 011  
continue;  
dfs(s.substr(i), cnt+1, comb + s.substr(0, i) + (cnt==3 ? ":" : ".") , res);  
}  
}
```

241. Different Ways to Add Parentheses(Medium) / 4/10

Given a string of numbers and operators, return all possible results from computing

Example 1:

Input: "2-1-1"

Output: [0, 2]

Explanation:

((2-1)-1) = 0

(2-(1-1)) = 2

Example 2:

Input: "2*3-4*5"

Output: [-34, -14, -10, -10, 10]

Explanation:

(2*(3-(4*5))) = -34

((2*3)-(4*5)) = -14

((2*(3-4))*5) = -10

(2*((3-4)*5)) = -10

(((2*3)-4)*5) = 10

**

思路

Code



C

```
class Solution {
public:
    vector<int> diffWaysToCompute(string input) {
        unordered_map<string, vector<int>> cache;
        return dfs_dp(input, cache);
    }
}
```

```
vector<int> dfs_dp(string input, unordered_map<string, vector<int>>& cache){
    vector<int> res;
    for (int i=0; i<input.size(); i++){
        char c = input[i];
        if (c == '+' || c == '-' || c == '*'){
            vector<int> left, right;
            string substr;
            substr = input.substr(0,i);
            if (cache.find(substr) != cache.end())
                left = cache[substr];
            else
                left = dfs_dp(substr, cache);
```

```

        substr = input.substr(i+1);
        if (cache.find(substr) != cache.end())
            right = cache[substr];
        else
            right = dfs_dp(substr, cache);

        // implement 笛卡爾積
        for (auto l : left){
            for (auto r : right){
                if (c == '+')    res.push_back( l+r );
                if (c == '-')    res.push_back( l-r );
                if (c == '*')    res.push_back( l*r );
            }
        }
    } // end of if op
}// end of string walk-through

// the input only consist integer, return int
if(res.empty())
    res.push_back(atoi(input.c_str()));
cache[input] = res;
return res;
}

};


```

```

## !95. Unique Binary Search Trees II(Medium) / 4/22

### 4/28 review(X)

```

```

C++
for(l:left){
for(r:right){
...
}
}

```

思路

如下圖，我們讓 $1 \sim n$ 的每個數當當看root，並回傳所有可能解
程式碼方面：
直覺的， i 是root，而 $start \sim i-1$ 是左子樹， $i+1 \sim end$ 是右子樹
而左右子樹都可以再分別遞迴下去做切割成左及右子樹
當無法再切割時($start > end$)時，就是NULL的狀態


```
! [] (assets/markdown-img-paste-20190629234109865.png)
! [] (assets/markdown-img-paste-20190629234128766.png)
! [] (assets/markdown-img-paste-20190629234149216.png)
```

Code

C++

/**

- Definition for a binary tree node.
- struct TreeNode {
- int val;
- TreeNode *left;
- TreeNode *right;
- TreeNode(int x) : val(x), left(NULL), right(NULL) {}
- };
/ class Solution { public: vector> generateTrees(int n) {
if (n == 0) return { };
return generateTreesDFS(1, n);
}

vector> generateTreesDFS(int start, int end){ vector> subtree;
if (start > end){
subtree.push_back(NULL);
}
else{
for (int i=start; i<= end; ++i){ vector> left = generateTreesDFS(start, i-1); vector> right =
generateTreesDFS(i+1, end);
for (auto l:left){
for (auto r:right){
TreeNode* node = new TreeNode(i); // generate the (sub)root
node -> left = l;
node -> right = r;
subtree.push_back(node);
}
}
}

```
    } // end of start-to-end walkthrough
}
return subtree;
}
};
```

!842. Split Array into Fibonacci Sequence(Medium) | 4/22

Given a **string** S of digits, such as S = "123456579", we can split it into a Fibonac

Formally, a Fibonacci-like sequence is a list F of non-negative integers such that:

```
0 <= F[i] <= 2^31 - 1, (that is, each integer fits a 32-bit signed integer type);
F.length >= 3;
and F[i] + F[i+1] = F[i+2] for all 0 <= i < F.length - 2.
Also, note that when splitting the string into pieces, each piece must not have ext
```

Return any Fibonacci-like sequence split from S, or return [] if it cannot be done.

Example 1:

Input: "123456579"

Output: [123, 456, 579]

Example 2:

Input: "11235813"

Output: [1, 1, 2, 3, 5, 8, 13]

Example 3:

Input: "112358130"

Output: []

Explanation: The task is impossible.

Example 4:

Input: "0123"

Output: []

Explanation: Leading zeroes are not allowed, so "01", "2", "3" is not valid.

Example 5:

Input: "1101111"

Output: [110, 1, 111]

Explanation: The output [11, 0, 11, 11] would also be accepted.

Note:

1 <= S.length <= 200

S contains only digits.

4/28 review(X)

思路

符合題意的數列其實可能不止一種，但是本題就讓返回一個就行了。不管返回幾個，總之不是求極值，DP在這裡就

現在來考慮遞歸函數的主體該怎麼寫，既然不知道要如何分割，那麼就要嘗試所有的情況，一個數字，兩個數字，

Code



```
C++
class Solution {
public:
vector<int> splitIntoFibonacci(string S) {
vector<int> out, res;
recursion(S, 0, out, res);
return res;
}

private:
void recursion(string S, int start, vector<int>& out, vector<int>& res) {
// as long as finding one solution, return
if (!res.empty()) return;
// out is more than three elements, and we walkthrough whole string, out is the result
if (out.size() >= 3 && start >= S.length()){
res = out; return;
}
for (int i=start; i < S.length() && s[i] != '0'); break;
long num = stol(s);
if (num >= INT_MAX) break;
// if it is not an Feb seq, note: we shouldn't break here, cuz the possib of 下個數字的與當前數
// 字的組合
if (out.size() >= 2 && num != (long)out[out.size() - 1] + out[out.size() - 2]) continue;
out.push_back(num);
recursion(S, i+1, out, res);
out.pop_back();
}
}
};

---
```

```
## !94. Binary Tree Inorder Traversal | 4/22
Given a binary tree, return the inorder traversal of its nodes' values.
```

Example:

Input: [1,null,2,3]



```
Output: [1,3,2]
Follow up: Recursive solution is trivial, could you do it iteratively?
### 4/28 review
```

```
### 思路
Pre-order: Duplicate binary tree
In-order: Binary search tree
Post-order: Delete binary tree
__Binary tree ~> DFS ~> stack__
=> __Travesal order == code order__
公式：
```

C++

```
while(p||!stack.empty()){
if(p){
stack.push(p);
// traversal order...
// ...
}
else{
p = stack.top(); stack.pop();
// traversal order...
// ...
}
}
```

空間複雜度更低的方法：Morris Traversal(不使用stack全部使用pointer)
Code

py

```
class Solution:
def inorderTraversal(self, root: TreeNode) -> List[int]:
stack = []
res = []
p = root
```

```
while p or stack:
    if p:
        stack.append(p)
        p = p.left
    else:
        top = stack.pop()
        res.append(top.val)
```

```
p = top.right  
return res
```

```
``` c++  
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 vector<int> inorderTraversal(TreeNode* root) {
 vector<int> res;
 stack<TreeNode*> stack;
 TreeNode* p = root;
 while(p || !stack.empty()){
 if(p){
 stack.push(p);
 p = p -> left;
 }
 else{
 p = stack.top(); stack.pop();
 res.push_back(p->val);
 p = p -> right;
 }
 }
 return res;
 }
};
```

---

## @144. Binary Tree Preorder Traversal | 4/22

---

Given a binary tree, return the preorder traversal of its nodes' values.

Example:

Input: [1,null,2,3]

```
1
\
2
```

/

3

Output: [1,2,3]

Follow up: Recursive solution is trivial, could you do it iteratively?

## 4/29 review

### 思路

children是vector型態直接用for迴圈走訪

與#589 的比較：589這題最大的不同是我們必須一次將所有的children全部放進stack裡，因此程式與輸出value的順序必須顛倒(因為stack的性質)

而#144: 一次push一個節點進stack，在下一次的迴圈我們希望輸出的value就是top，因此程式與輸出value的順序要一樣

### Code

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 vector<int> preorderTraversal(TreeNode* root) {
 vector<int> res;
 stack<TreeNode*> stack;
 TreeNode* p = root;
 while (!stack.empty() || p){
 if (p){
 stack.push(p);
 res.push_back(p->val);
 p = p->left;
 }
 else{
 p = stack.top(); stack.pop();
 p = p->right;
 }
 }
 return res;
 }
};
```

## @145. Binary Tree Postorder Traversal | 4/22

Given a binary tree, return the postorder traversal of its nodes' values.

Example:

Input: [1,null,2,3]

```
1
 \
2
/
3
```

Output: [3,2,1]

Follow up: Recursive solution is trivial, could you do it iteratively?

### 4/29 review

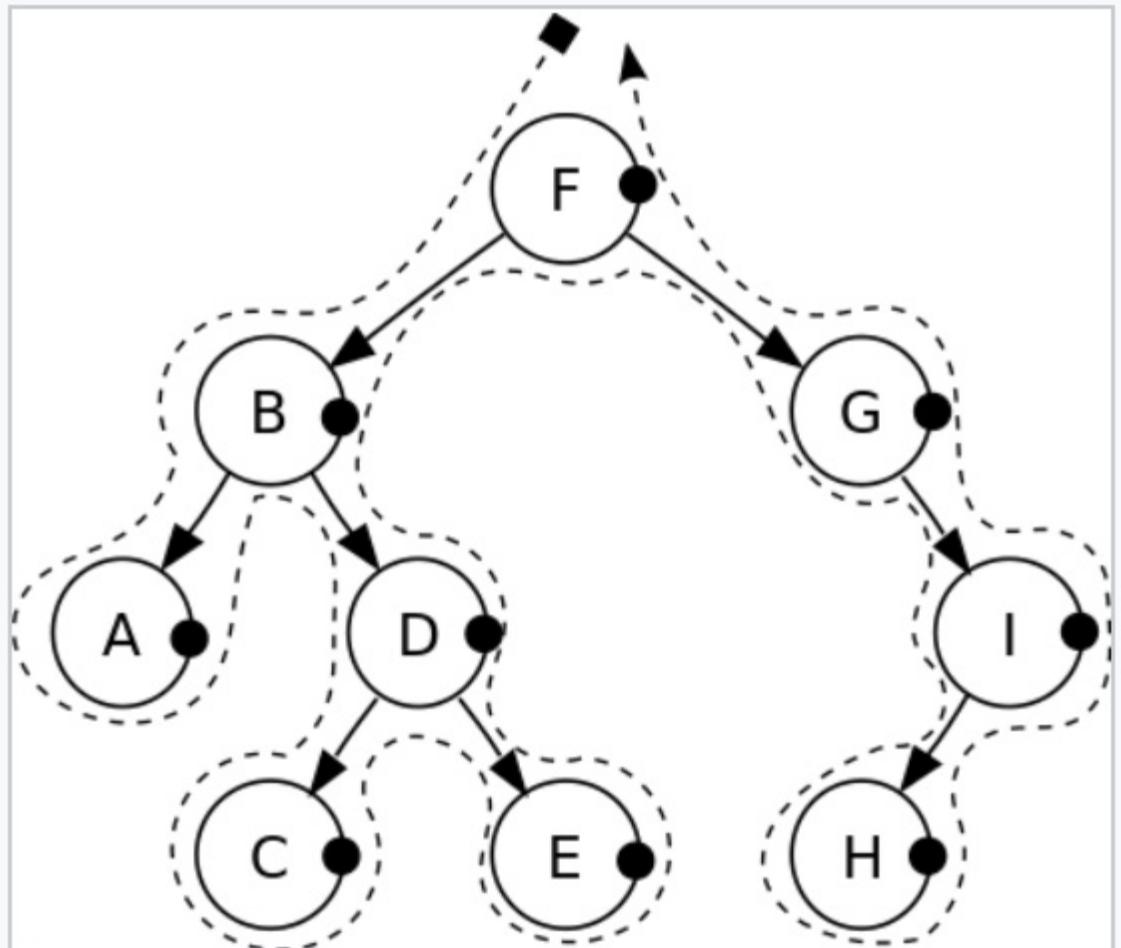
#### 思路

post traversal order: left->right->root

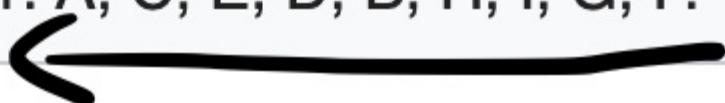
can't directly code it cuz there are no pointer p in the else statement

hence, reverse the traversal order to [root->right->left]

and also reverse the push\_back order in vector ( insert from begin )



Post-order: A, C, E, D, B, H, I, G, F.



insert order

## Code

```

class Solution:
 def preorderTraversal(self, root: TreeNode) -> List[int]:
 stack = []
 res = []
 p = root

 while p or stack:
 if p:
 stack.append(p)
 res.append(p.val)
 p = p.left
 else:
 top = stack.pop()

```

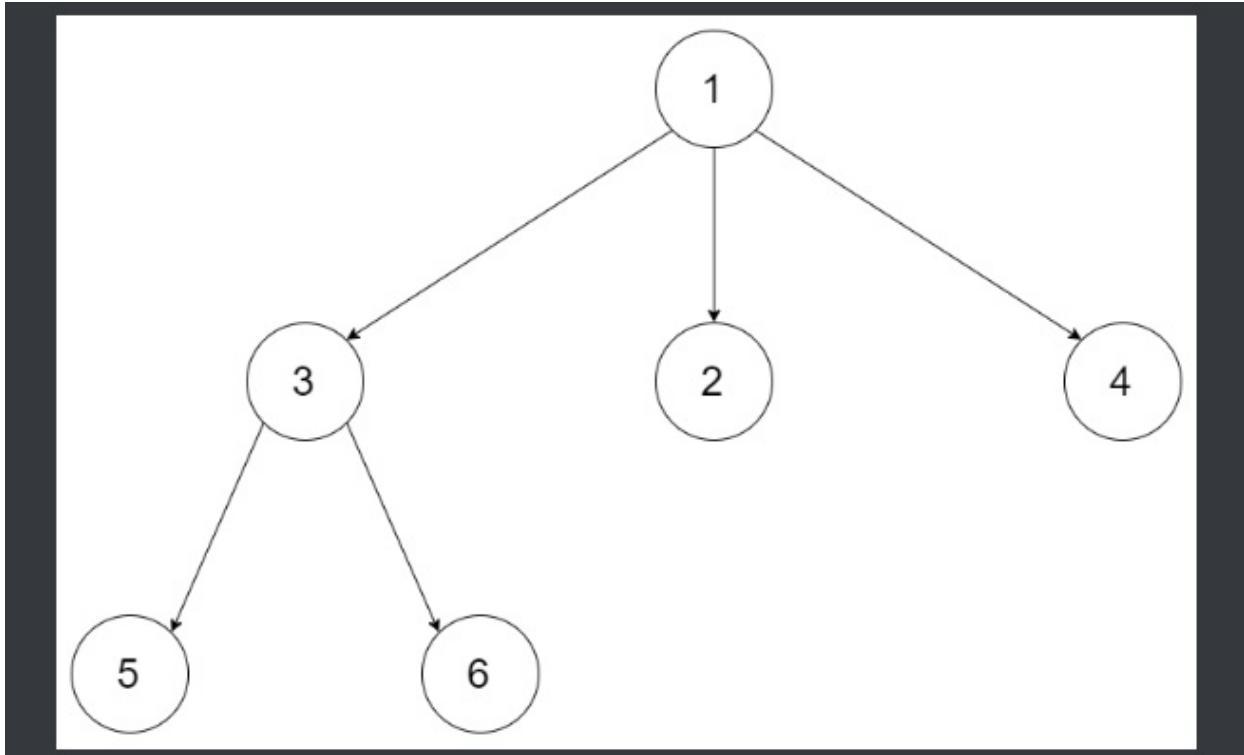
```
 p = top.right
return res
```

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
/*
traversal order: left->right->root
can't directly code it cuz the pointer will lose his parents
hence, reverse the traversal order to [root->right->left]
and also reverse the push_back order in vector (insert from begin)
*/
class Solution {
public:
 vector<int> postorderTraversal(TreeNode* root) {
 vector<int> res;
 stack<TreeNode*> stack;
 TreeNode* p = root;
 while(p || !stack.empty()){
 if(p){
 stack.push(p);
 res.insert(res.begin(), p->val);
 p = p->right;
 }
 else{
 p = stack.top(); stack.pop();
 p = p->left;
 }
 }
 return res;
 }
};
```

## !589. N-ary Tree Preorder Traversal | 4/23

Given an n-ary tree, return the preorder traversal of its nodes' values.

For example, given a 3-ary tree:



Return its preorder traversal as: [1,3,5,6,2,4].

## 4/29 review

### 思路

children是vector型態直接用for迴圈走訪

與#144 的比較：這題最大的不同是我們必須一次將所有的children全部放進stack裡，因此程式與輸出value的順序必須顛倒(因為stack的性質)

而#144: 一次push一個節點進stack，在下一次的迴圈我們希望輸出的value就是top，因此程式與輸出value的順序要一樣

### Code

```

/*
// Definition for a Node.
class Node {
public:
 int val;
 vector<Node*> children;

 Node() {}

 Node(int _val, vector<Node*> _children) {
 val = _val;
 children = _children;
 }
};
 */

```

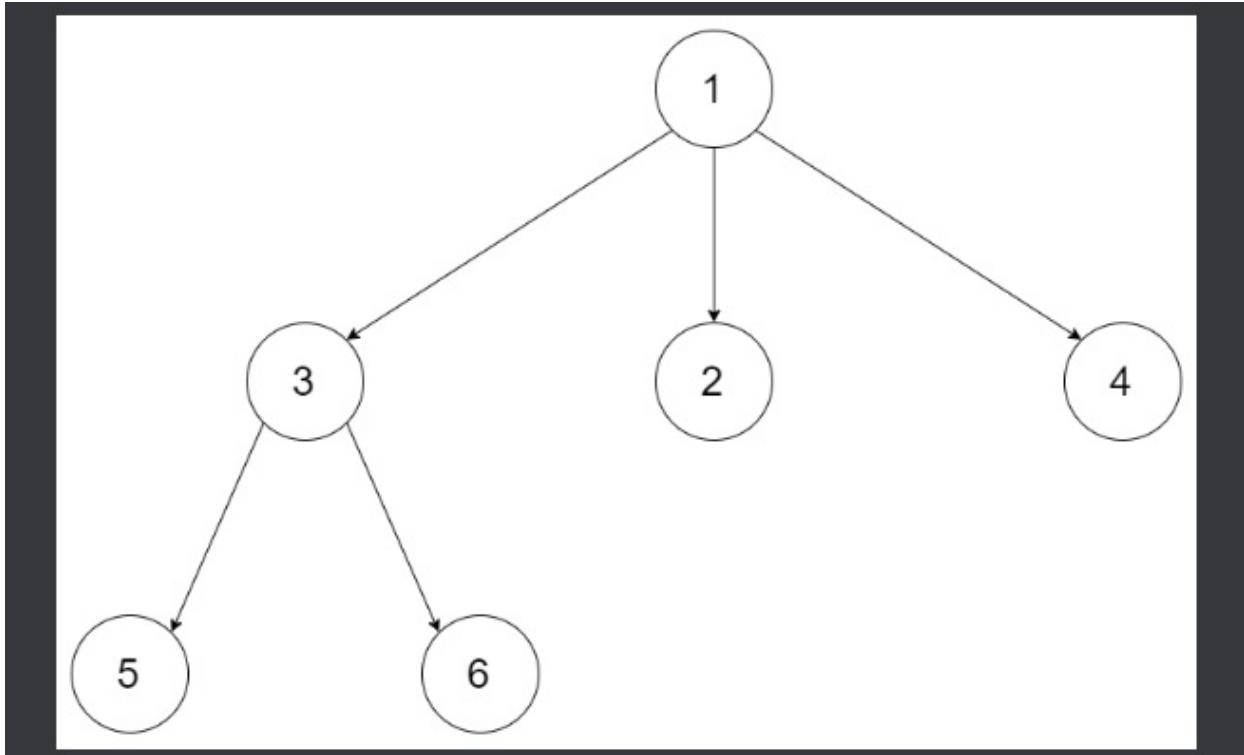
```
 children = _children;
 }
};

/*
// root, left, right
class Solution {
public:
 vector<int> preorder(Node* root) {
 if (!root) return { };
 vector<int> res;
 stack<Node*> stack;
 stack.push(root);
 while(!stack.empty()){
 Node* p = stack.top(); stack.pop();
 res.push_back(p->val);
 // suppose the children vector order is left and right
 // with the stack, should push right first cuz we want left out first
 for(int i=p->children.size()-1; i>=0; i--){
 stack.push(p->children[i]);
 }
 }
 return res;
 }
};
```

## @590. N-ary Tree Postorder Traversal | 4/23

Given an n-ary tree, return the postorder traversal of its nodes' values.

For example, given a 3-ary tree:



Return its postorder traversal as: [5,6,3,2,4,1].

Note:

Recursive solution is trivial, could you do it iteratively?

4/29 review

思路

children是vector型態直接用for迴圈走訪即可  
post order使用反轉手法

Code

```

/*
// Definition for a Node.
class Node {
public:
 int val;
 vector<Node*> children;

 Node() {}

 Node(int _val, vector<Node*> _children) {
 val = _val;
 children = _children;
 }
};

class Solution {
public:
 vector<int> postorder(Node* root) {
 if (!root) return {};
 stack<Node*> s;
 s.push(root);
 vector<int> res;
 while (!s.empty()) {
 Node* cur = s.top();
 s.pop();
 res.push_back(cur->val);
 for (auto child : cur->children) {
 s.push(child);
 }
 }
 reverse(res.begin(), res.end());
 return res;
 }
};

```

```

 val = _val;
 children = _children;
}
};

*/
// left, right, root ->(reverse)-> root, right, left
//
class Solution {
public:
 vector<int> postorder(Node* root) {
 if(!root) return { };
 vector<int> res;
 stack<Node*> stack;
 stack.push(root);
 while (!stack.empty()){
 Node* p = stack.top(); stack.pop();
 res.insert(res.begin(), p->val); // could use push_back and reverse the
 // want right come-out first so left first in
 for (int i=0; i< p->children.size(); i++){
 stack.push(p->children[i]);
 }
 }
 return res;
 }
};

```

## @100. Same Tree | 4/23

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

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

### Example 1:

**Input:**      1                1  
              / \            / \\\  
              2    3            2    3  
  
[1,2,3],     [1,2,3]

**Output:** true

### Example 2:

**Input:**      1                1  
              /                \\\  
              2                2  
  
[1,2],        [1,null,2]

**Output:** false

### Example 3:

**Input:**      1                1  
              / \            / \\\  
              2    1            1    2  
  
[1,2,1],     [1,1,2]

**Output:** false

4/29 review

思路

iterative的手法，前中後序接可以應用來比較

Code

Recursive

```

class Solution {
public:
 bool isSameTree(TreeNode *p, TreeNode *q) {
 if (!p && !q) return true;
 if ((p && !q) || (!p && q) || (p->val != q->val)) return false;
 return isSameTree(p->left, q->left) && isSameTree(p->right, q->right);
 }
};

```

Iterative(faster)

```

/*
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 bool isSameTree(TreeNode* p, TreeNode* q) {
 stack<TreeNode*> stack;
 stack.push(p), stack.push(q);
 while(!stack.empty()){
 q = stack.top(); stack.pop();
 p = stack.top(); stack.pop();
 if (!p && !q) continue;
 if ((!p && q) || (p && !q) || (p->val != q->val)) return false;
 // 在前面兩個if完全過濾掉p的child有NULL的情形了，所以這邊才能放心push Note: stack
 stack.push(p->right), stack.push(q->right);
 stack.push(p->left), stack.push(q->left);
 }
 return true;
 }
};
```

## @101. Symmetric Tree | 4/23

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:

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

But the following [1,2,2,null,3,null,3] is not:

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

## 思路

切割成左右子樹比較

直覺上使用queue來處理

實測:Queue的速度比Stack快很多

## Code

Recursive

```
class Solution {
public:
 bool isSymmetric(TreeNode* root) {
 if (!root) return true;
 return isSymmetric(root->left, root->right);
 }
 bool isSymmetric(TreeNode* left, TreeNode* right) {
 if (!left && !right) return true;
 if (left && !right || !left && right || left->val != right->val) return false;
 return isSymmetric(left->left, right->right) && isSymmetric(left->right, right->left);
 }
};
```

Iterative

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 bool isSymmetric(TreeNode* root) {
 if(!root) return true;
 TreeNode* left = root->left;
 TreeNode* right = root->right;
 queue<TreeNode*> queue;
 queue.push(left), queue.push(right);
 while(!queue.empty()){
 left = queue.front(); queue.pop();
 right = queue.front(); queue.pop();
 if (!left && !right) continue;
 if ((!left && right) || (left &&!right) || (left->val != right->val))
 queue.push(left->left), queue.push(right->right);
 queue.push(left->right),queue.push(right->left);
 }
 return true;
 }
};

```

## @104. Maximum Depth of Binary Tree | 4/23

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.

Note: A leaf is a node with no children.

Example:

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

```
 3
 / \
 9 20
 / \
 15 7
```

return its depth = 3.

return its depth = 3

## 4/29 review

### 思路

使用層序遍歷二叉樹，然後計數總層數，即為二叉樹的最大深度

### Code

#### Recursive

```
class Solution {
public:
 int maxDepth(TreeNode* root) {
 if (!root) return 0;
 return 1 + max(maxDepth(root->left), maxDepth(root->right));
 }
};
```

#### Iterative

```
class Solution {
public:
 int maxDepth(TreeNode* root) {
 if(!root)
 return 0;
 int res = 0;
 // We need to consider all the node in the layer, hence BFS
 queue<TreeNode*> queue;
 queue.push(root);
 while (!queue.empty()){
 ++res;
 // shouldn't inc counter if in the same layer, hence record their children
 for (int i=0, n=queue.size(); i<n; ++i){
```

```

 // for (int i=0, i<queue.size(); ++i) errors happen cuz the inc of queue.
 // for (int i=queue.size(); i>0; i--) THIS would be the safestest
 TreeNode* t = queue.front();
 queue.pop();
 if(t->left)
 queue.push(t->left);
 if(t->right)
 queue.push(t->right);
 }
}
return res;
}
};


```

## \*110. Balanced Binary Tree | 4/24

4/29 review

思路

Code

Solution1: calculate all the nodes

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 bool isBalanced(TreeNode* root) {
 if (!root) return true;
 if (abs(getDepth(root->left) - getDepth(root->right)) > 1) return false;
 // need to check every node in this tree is all balanced tree, hence recurs
 return isBalanced(root->left) && isBalanced(root->right);
 }

 int getDepth(TreeNode* root){
 if (!root) return 0;

```

```
 return 1 + max(getDepth(root->left), getDepth(root->right));
 }
};
```

Solution2 做了一些計算上的優化

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 bool isBalanced(TreeNode* root) {
 if (checkDepth(root) == -1) return false;
 return true;
 }
 int checkDepth(TreeNode* root){
 if (!root) return 0; // the depth of leaf child is zero
 // 只要一發現左右子樹有不平衡的狀況，直接返回-1，省去計算深度的步驟
 int left = checkDepth(root->left);
 if (left == -1) return -1;
 int right = checkDepth(root->right);
 if (right == -1) return -1;
 // 精髓在此，把計算深度的過程放在最後面，只有當左右子樹都是平衡的情況下才有計算具體深度的。
 if (abs(left-right) > 1) return -1;
 return 1 + max(left, right);
 }
};
```

## @111. Minimum Depth of Binary Tree | 4/24

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

Note: A leaf is a node with no children.

Example:

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



return its minimum depth = 2.

return its minimum depth = 2.

4/29 review

## 思路

Iterative的方法還是比Recursive快上不少

我們迭代來做，層序遍歷，記錄遍歷的層數，一旦我們遍歷到第一個葉結點，就將當前層數返回，即為二叉樹的最小深度

## Code

Recursive

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * };
 */
int minDepth(TreeNode* root) {
 if (root == NULL) return 0;
 if (root->left == NULL && root->right == NULL) return 1;
 int leftDepth = minDepth(root->left);
 int rightDepth = minDepth(root->right);
 if (leftDepth < rightDepth) return leftDepth + 1;
 else return rightDepth + 1;
}
```

```

* TreeNode *right;
* TreeNode(int x) : val(x), left(NULL), right(NULL) {}
* };
*/
class Solution {
public:
 int minDepth(TreeNode* root) {
 if (!root) return 0;
 if (!root->left) return 1+minDepth(root->right);
 if (!root->right) return 1+minDepth(root->left);
 return 1 + min(minDepth(root->left), minDepth(root->right));
 }
};

```

## Iterative

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 int minDepth(TreeNode* root) {
 if (!root) return 0;
 int res = 0;
 queue<TreeNode*> queue;
 queue.push(root);
 while (!queue.empty()){
 res++;
 for (int i=queue.size(); i>0; i--){
 TreeNode* p = queue.front(); queue.pop();
 if (!p->left && !p->right) return res; // in BFS, the first leaf w
 if (p->left) queue.push(p->left);
 if (p->right) queue.push(p->right);
 }
 }
 return -1;
 }
};

```

Given two non-empty binary trees s and t, check whether tree t has exactly the same structure and node values with a subtree of s. A subtree of s is a tree consists of a node in s and all of this node's descendants. The tree s could also be considered as a subtree of itself.

**Example 1:**

Given tree s:

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

Given tree t:

```
 4
 / \
 1 2
```

Return **true**, because t has the same structure and node values with a subtree of s.

**Example 2:**

Given tree s:

```
 3
 / \
 4 5
 / \
1 2
 /
0
```

Given tree t:

```
 4
 / \
 1 2
```

Return **false**.

# 思路

## Code

Recursive

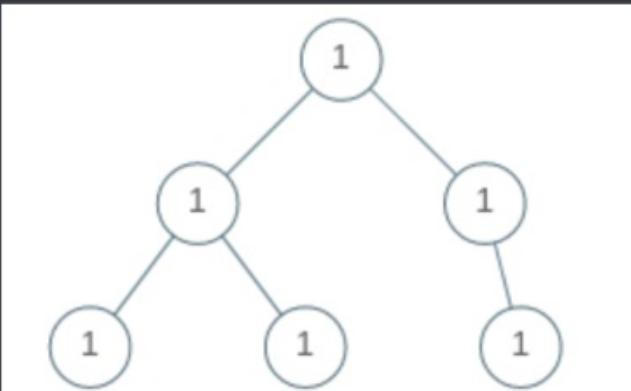
```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 bool isSubtree(TreeNode* s, TreeNode* t) {
 if (!s) return false;
 if (isSame(s, t)) return true;
 return isSubtree(s->left, t) || isSubtree(s->right, t);
 }
 bool isSame(TreeNode* s, TreeNode* t){
 if (!s && !t) return true;
 if ((!s && t) || (s && !t)) return false;
 if (s->val != t->val) return false;
 return isSame(s->left, t->left) && isSame(s->right, t->right);
 }
};
```

## @965. Univalued Binary Tree | 4/24

A binary tree is univalued if every node in the tree has the same value.

Return true if and only if the given tree is univalued.

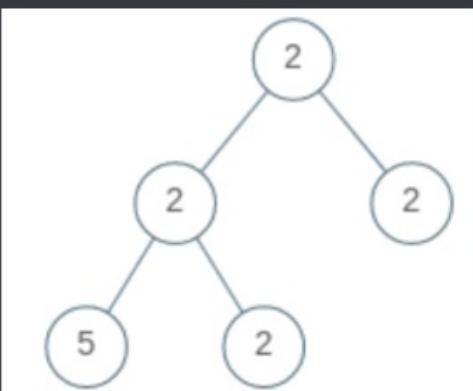
### Example 1:



**Input:** [1,1,1,1,1,null,1]

**Output:** true

### Example 2:



**Input:** [2,2,2,5,2]

**Output:** false

思路

Code

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
```

```

/*
class Solution {
public:
 bool isUnivalTree(TreeNode* root) {
 if (!root) return true;
 if (root->left && root->val != root->left->val) return false;
 if (root->right && root->val != root->right->val) return false;
 return isUnivalTree(root->left) && isUnivalTree(root->right);
 }
};

```

## @102. Binary Tree Level Order Traversal | 4/25

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],



return its level order traversal as:

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

## 思路

Level order BFS

## Code

## Iterative

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 vector<vector<int>> levelOrder(TreeNode* root) {
 if (!root) return {};
 vector<vector<int>> res;
 queue<TreeNode*> queue;
 queue.push(root);
 while(!queue.empty()){
 vector<int> thisLevel;
 for (int i=queue.size(); i>0; i--){
 TreeNode* p = queue.front(); queue.pop();
 thisLevel.push_back(p->val);

 if(p->left) queue.push(p->left);
 if(p->right) queue.push(p->right);
 }
 res.push_back(thisLevel);
 }
 return res;
 }
};
```

## Recursive:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 vector<vector<int>> levelOrder(TreeNode* root) {
 vector<vector<int>> res;
 recursion(root, 0, res);
 return res;
 }
};
```

```

private:
 void recursion(TreeNode* root, int level, vector<vector<int>>& res){
 if (!root) return;
 if (res.size() == level) res.push_back({}); // give new level a room
 res[level].push_back(root->val);
 recursion(root->left, level+1, res);
 recursion(root->right, level+1, res);
 }
};

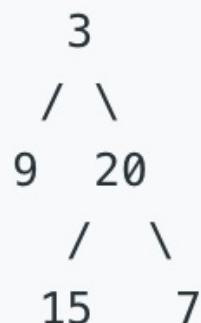
```

## @107. Binary Tree Level Order Traversal II | 4/25

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],



return its bottom-up level order traversal as:

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

## 思路

## Code

Iterative:

```
// 參考 #102 將push_back改成insert
```

Recursive:

```
// 參考 #102 在return res之前將vector reverse
```

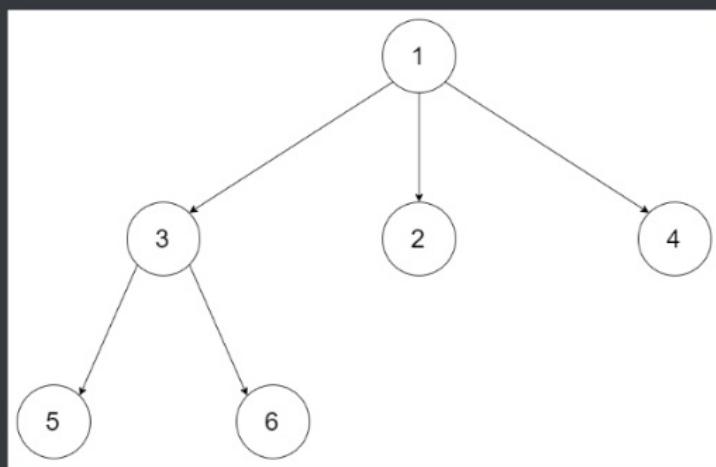
---

## @429. N-ary Tree Level Order Traversal | 4/25

---

Given an n-ary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

For example, given a 3-ary tree:



We should return its level order traversal:

```
[
 [1],
 [3,2,4],
 [5,6]
]
```

## 思路

## Code

```
/*
// Definition for a Node.
class Node {
public:
 int val;
 vector<Node*> children;

 Node() {}

 Node(int _val, vector<Node*> _children) {
 val = _val;
 children = _children;
 }
};
*/
class Solution {
```

```

public:
 vector<vector<int>> levelOrder(Node* root) {
 if(!root) return {};
 vector<vector<int>> res;
 queue<Node*> queue;
 queue.push(root);
 while (!queue.empty()){
 vector<int> oneLevel;
 for (int i=queue.size(); i>0; --i){
 root = queue.front(); queue.pop();
 oneLevel.push_back(root->val);
 if (!root->children.empty()){
 for (auto child : root->children) queue.push(child);
 }
 }
 res.push_back(oneLevel);
 }
 return res;
 }
};

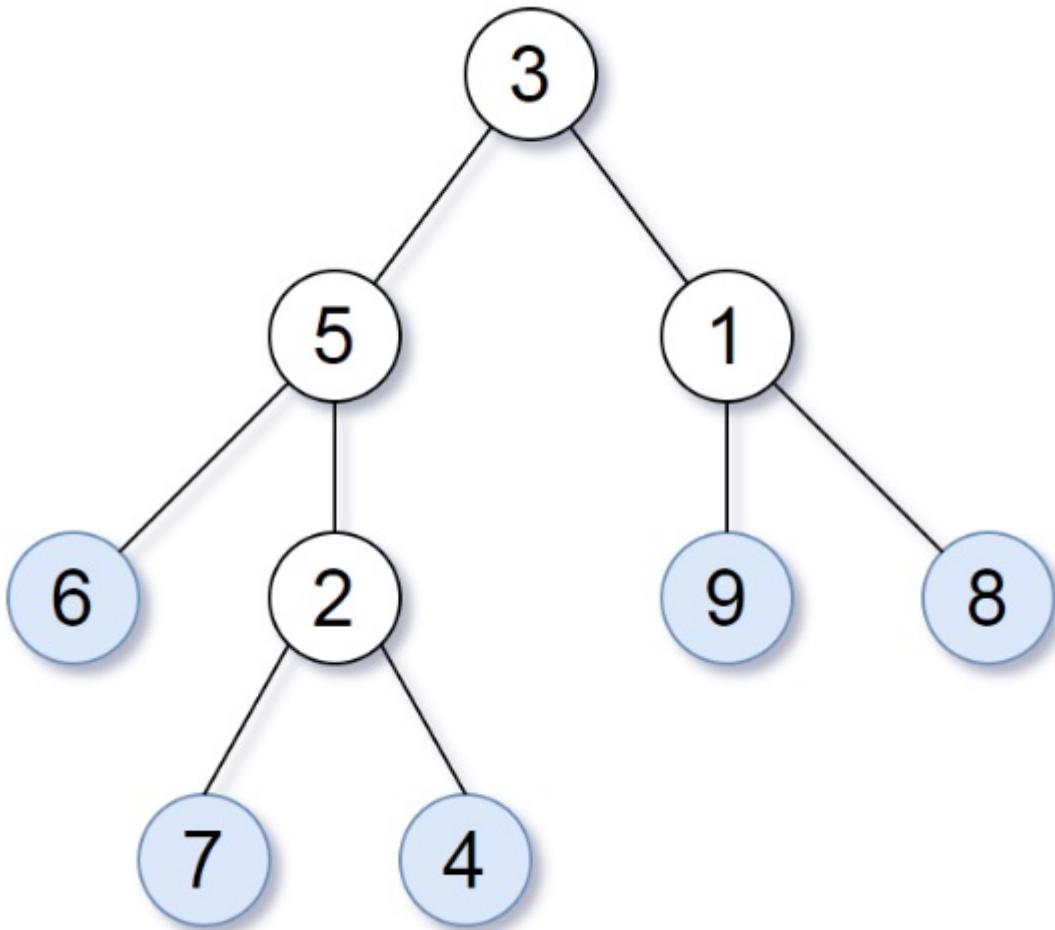
```

---

## @872. Leaf-Similar Trees | 4/25

---

Consider all the leaves of a binary tree. From left to right order, the values of those leaves form a leaf value sequence.



For example, in the given tree above, the leaf value sequence is (6, 7, 4, 9, 8).

Two binary trees are considered leaf-similar if their leaf value sequence is the same.

Return true if and only if the two given trees with head nodes root1 and root2 are leaf-similar.

## 思路

## Code

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:

```

```

bool leafSimilar(TreeNode* root1, TreeNode* root2) {
 string s1, s2;
 dfs(root1, s1);
 dfs(root2, s2);
 return s1==s2;
}

private:
 void dfs(TreeNode* root, string& seq){
 if (!root) return;
 if (!root->left && !root->right) seq += to_string(root->val) + "# ";
 else{
 dfs(root->left, seq);
 dfs(root->right, seq);
 }
 }
};

```

## \*987. Vertical Order Traversal of a Binary Tree | 4/25

---

Given a binary tree, return the vertical order traversal of its nodes values.

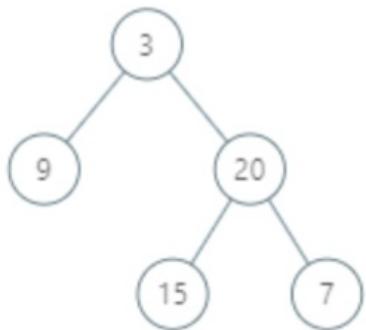
For each node at position (X, Y), its left and right children respectively will be at positions (X-1, Y-1) and (X+1, Y-1).

Running a vertical line from X = -infinity to X = +infinity, whenever the vertical line touches some nodes, we report the values of the nodes in order from top to bottom (decreasing Y coordinates).

If two nodes have the same position, then the value of the node that is reported first is the value that is smaller.

Return a list of non-empty reports in order of X coordinate. Every report will have a list of values of nodes.

**Example 1:**



**Input:** [3,9,20,null,null,15,7]

**Output:** [[9],[3,15],[20],[7]]

**Explanation:**

Without loss of generality, we can assume the root node is at position (0, 0):

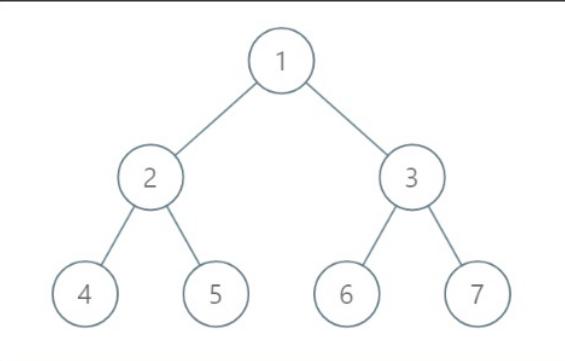
Then, the node with value 9 occurs at position (-1, -1);

The nodes with values 3 and 15 occur at positions (0, 0) and (0, -2);

The node with value 20 occurs at position (1, -1);

The node with value 7 occurs at position (2, -2).

## Example 2:



**Input:** [1,2,3,4,5,6,7]

**Output:** [[4],[2],[1,5,6],[3],[7]]

### Explanation:

The node with value 5 and the node with value 6 have the same position according to the given scheme.

However, in the report "[1,5,6]", the node value of 5 comes first since 5 is smaller than 6.

## 5/6 review

採用第二種寫法

學習到：

1. struct 重新賦予x, y值的手法
2. for迴圈走訪map的pair手法
3. append a vector to another vector的手法(vA.back().insert(vA.back().end(), vB.begin(), vB.end()))

## 思路

利用map去紀錄所有資料

## Code

直覺的寫法

```
/**
 * Definition for a binary tree node.
 */
```

```

* struct TreeNode {
* int val;
* TreeNode *left;
* TreeNode *right;
* TreeNode(int x) : val(x), left(NULL), right(NULL) {}
* };
*/
class Solution {
public:
 vector<vector<int>> verticalTraversal(TreeNode* root) {
 vector<vector<int>> ans;
 int h_dist = 0, v_dist = 0;
 queue< pair<TreeNode*, pair<int, int>> q; // queue of Node and (x,y)
 map<int, vector<pair<int, int>>> outMap; // map of output order map<h_dist
 q.push(make_pair(root, make_pair(h_dist, v_dist))); // pushinto queue & ini
 while(!q.empty()){
 /* update the map */
 auto front = q.front(); q.pop();
 auto node = front.first; //get node
 auto curDis = front.second; // get posi
 h_dist = curDis.first;
 v_dist = curDis.second;

 outMap[h_dist].push_back(make_pair(v_dist, node->val)); // update the o

 /* update the queue */
 if (node->left) q.push(make_pair(node->left, make_pair(h_dist-1, v_dis
 if (node->right) q.push(make_pair(node->right, make_pair(h_dist+1, v_d
 }

 ans.resize(outMap.size()); // give our answer vector enough space
 int index=0;

 for (auto elem: outMap){ // each round is the elem with same h_dist
 auto row_vec = elem.second; // get the vector(v_dsit, val) in outMa
 sort(row_vec.begin(), row_vec.end()); // sort by the v_dist (if same, t

 for (auto value: row_vec){
 ans[index].push_back(value.second);
 }
 index++;
 }

 return ans;
 }
 };
}

```

利用structure 以及 map  
insert from back = push\_back

```
class Solution {
```

```

struct Node {
 TreeNode* n;
 int x, y;
};

public:
 vector<vector<int>> verticalTraversal(TreeNode* root) {
 vector<vector<int>> ans;
 map<int, map<int, vector<int>>> mp;
 queue<Node*> q;
 q.push(new Node{root, 0, 0});
 while(!q.empty()) {
 Node* node = q.front();
 q.pop();
 if(node->n->left) q.push(new Node{node->n->left, node->x - 1, node->y + 1});
 if(node->n->right) q.push(new Node{node->n->right, node->x + 1, node->y + 1});
 mp[node->x][node->y].push_back(node->n->val);
 }
 for(pair<int, map<int, vector<int>>> p1 : mp) {
 ans.push_back({}); // assign a room from back
 for(pair<int, vector<int>> p2 : p1.second) {
 vector<int> v = p2.second; // vector with all node which has same (x,y)
 sort(v.begin(), v.end());
 ans.back().insert(ans.back().end(), v.begin(), v.end());
 // ans 的最後面.插入(從ans最後面的最後面插入, value.begin()~end());
 // (即是前面的room)
 }
 }
 return ans;
 }
};

```

## \*814. Binary Tree Pruning | 4/30

We are given the head node root of a binary tree, where additionally every node's value is either a 0 or a 1.

Return the same tree where every subtree (of the given tree) not containing a 1 has been removed.

(Recall that the subtree of a node X is X, plus every node that is a descendant of X.)

### Example 1:

**Input:** [1,null,0,0,1]

**Output:** [1,null,0,null,1]

### Explanation:

Only the red nodes satisfy the property "every subtree not containing a 1".

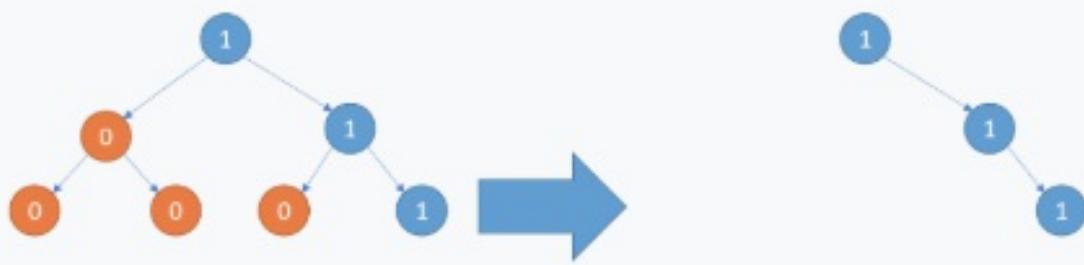
The diagram on the right represents the answer.



### Example 2:

Input: [1,0,1,0,0,0,1]

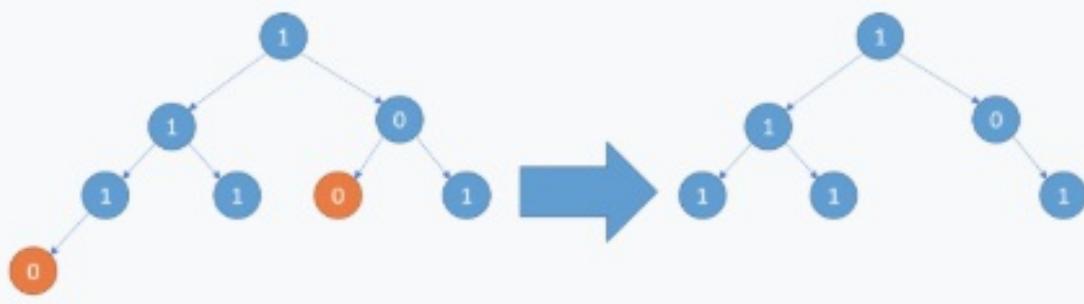
Output: [1,null,1,null,1]



### Example 3:

Input: [1,1,0,1,1,0,1,0]

Output: [1,1,0,1,1,null,1]



## 思路

對於例子2來說，如果移除了第三行的3個葉結點後，那麼第二行的那個值為0的結點也變成了葉

結點，繼續移除即可，所以與其找值全為0的子樹，我們可以不斷的移除值為0的葉結點，全都移除後那麼值全為0的子樹也就都被移除了。

對於玩二叉樹的題，十有八九都是用遞歸，所以我們應該首先就考慮遞歸的解法，然後再想按什麼順序來遍歷二叉樹呢？層序，先序，中序，還是後序？根據這道題的特點，我們要從末尾來一層一層的移除值為0的葉結點，所以天然時候用後序遍歷。分別對左右子結點調用遞歸函數，此時判斷，如果當前結點是值為0的葉結點，那麼移除該結點，即返回空，否則返回原結點即可

## Code

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 TreeNode* pruneTree(TreeNode* root) {
 if (!root) return NULL;
 root->left = pruneTree(root->left);
 root->right = pruneTree(root->right);
 return (!root->left && !root->right && root->val == 0)? NULL:root;
 }
};
```

## \*669. Trim a Binary Search Tree | 4/30

Given a binary search tree and the lowest and highest boundaries as L and R, trim the tree so that all its elements lies in [L, R] ( $R \geq L$ ). You might need to change the root of the tree, so the result should return the new root of the trimmed binary search tree.

### Example 1:

#### Input:

```
1
/
0 2
```

```
L = 1
R = 2
```

#### Output:

```
1
 \
 2
```

### Example 2:

#### Input:

```
3
/
0 4
 \
 2
 /
1
```

```
L = 1
R = 3
```

#### Output:

```
3
/
2
/
1
```

## 思路

正確方法其實應該是在遍歷的過程中就修改二叉樹，移除不合題意的結點。當然對於二叉樹的

題，十有八九都是要用遞歸來解的。首先判斷如果root為空，那麼直接返回空即可。然後就是要看根結點是否在範圍內，如果根結點值小於L，那麼返回對其右子結點調用遞歸函數的值；如果根結點大於R，那麼返回對其左子結點調用遞歸函數的值。如果根結點在範圍內，將其左子結點更新為對其左子結點調用遞歸函數的返回值，同樣，將其右子結點更新為對其右子結點調用遞歸函數的返回值。最後返回root即可

## Code

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 TreeNode* trimBST(TreeNode* root, int L, int R) {
 if (!root) return NULL;
 root->left = trimBST(root->left, L, R); // 遞迴走訪到葉節點
 root->right = trimBST(root->right, L, R);
 if (root->val < L) return trimBST(root->right, L, R); // 修正節點，如果小於mi
 if (root->val > R) return trimBST(root->left, L, R);
 return root;
 }
};
```

## \*112. Path Sum | 4/30

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.

Note: A leaf is a node with no children.

### 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.

## 思路

## Code

Recursive:

```
/**
 * Definition for a binary tree node.
 */
struct TreeNode {
 int val;
 TreeNode *left;
 TreeNode *right;
 TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

class Solution {
public:
 bool hasPathSum(TreeNode* root, int sum) {
 if (!root) return false;
 if (!root->left && !root->right && root->val == sum) return true;
 return hasPathSum(root->left, sum - root->val) || hasPathSum(root->right, s
 }
};
```

Iterative:

## @113. Path Sum II | 4/30

Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals the given sum.

Note: A leaf is a node with no children.

### Example:

Given the below binary tree and `sum = 22`,



Return:

```
[
 [5,4,11,2],
 [5,8,4,5]
]
```

## 思路

找所有解，使用遞迴就對了！

## Code

Recursive:

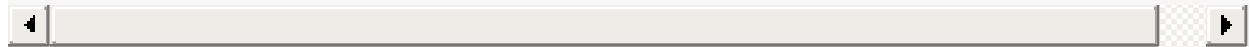
```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
```

```

* TreeNode *right;
* TreeNode(int x) : val(x), left(NULL), right(NULL) {}
* };
*/
class Solution {
public:
 vector<vector<int>> pathSum(TreeNode* root, int sum) {
 vector<vector<int>> res;
 vector<int> comb;
 dfs(root, sum, res, comb);
 return res;
 }
 void dfs(TreeNode* root, int sum, vector<vector<int>>& res, vector<int>& comb){
 if (!root) return;
 comb.push_back(root->val);

 if (!root->left && !root->right && root->val == sum){
 res.push_back(comb);
 /*comb.pop_back(); */
 //return; 這裡不能return! 會少了pop_back()，加上面那一行就會過
 }
 dfs(root->left, sum - root->val, res, comb);
 dfs(root->right, sum - root->val, res, comb);
 comb.pop_back();
 }
};

```



Iterative:

## \*437. Path Sum III | 5/1

You are given a binary tree in which each node contains an integer value.

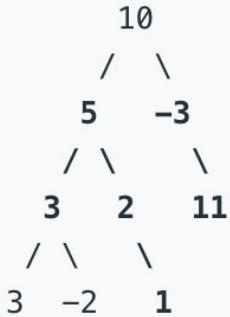
Find the number of paths that sum to a given value.

The path does not need to start or end at the root or a leaf, but it must go downwards (traveling only from parent nodes to child nodes).

The tree has no more than 1,000 nodes and the values are in the range -1,000,000 to 1,000,000.

### Example:

```
root = [10,5,-3,3,2,null,11,3,-2,null,1], sum = 8
```



Return 3. The paths that sum to 8 are:

1. 5 → 3
2. 5 → 2 → 1
3. -3 → 11

## 思路

我們可以用遞歸來做，相當於先序遍歷二叉樹，對於每一個節點都有記錄了一條從根節點到當前節點到路徑，同時用一個變量curSum記錄路徑節點總和，然後我們看curSum和sum是否相等，相等的話結果res加1，不等的話我們來繼續查看子路徑和有沒有滿足題意的，做法就是每次去掉一個節點，看路徑和是否等於給定值，注意最後必須留一個節點，不能全去掉了，因為如果全去掉了，路徑之和為0，而如果假如給定值剛好為0的話就會有問題  
key: 維護一個從root到curNode的vector

## 5/6review

錯誤：在寫的時候並沒有使用curSum去紀錄，而是直接對sum去做扣值  
直接對sum去做扣值並沒有辦法去進行每次去掉一個節點的動作

## Code

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
```

```

* TreeNode *right;
* TreeNode(int x) : val(x), left(NULL), right(NULL) {}
* };
*/
class Solution {
public:
 int pathSum(TreeNode* root, int sum) {
 vector<TreeNode*> path;
 int cnter = 0;
 recur(root, sum, 0, cnter, path);
 return cnter;
 }

 void recur(TreeNode* node, int sum, int curSum, int& cnter, vector<TreeNode*>&
 if (!node) return;
 curSum += node->val;
 path.push_back(node); // construct a path from root to leaf
 if (curSum == sum) cnter++;
 // check the curSum == sum if we remove some node in past path
 int temp = curSum; // we don't want to change the curSum, make a copy for i
 for (int i=0; i < path.size() -1; i++){ // we shouldn't remove current nod
 temp -= path[i]->val;
 if (temp == sum) cnter++;
 }
 // recursion
 recur(node->left, sum, curSum, cnter, path);
 recur(node->right, sum, curSum, cnter, path);
 path.pop_back();
}
};


```

## @543. Diameter of Binary Tree | 5/1

---

Given a binary tree, you need to compute the length of the diameter of the tree. The diameter of a binary tree is the length of the **longest** path between any two nodes in a tree. This path may or may not pass through the root.

### Example:

Given a binary tree



Return **3**, which is the length of the path [4,2,1,3] or [5,2,1,3].

**Note:** The length of path between two nodes is represented by the number of edges between them.

## 思路

拆解題目後：求根結點的左右兩個子樹的深度之和  
但題目並不限定一定要通過root，  
因此我們可以直接參考方法2，  
在計算深度時同時紀錄最深的節點，  
透過方法3的hashmap可以去除重複計算的情形。

可以從方法1歸納出：當不限於root時，就是recursive main function with root->left and root->right

## Code

多此一舉用了兩個recursive，過不了測資

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 }
```

```

 * };
 */
class Solution {
public:
 int diameterOfBinaryTree(TreeNode* root) {
 if(!root) return 0;
 int res = getDeep(root->left) + getDeep(root->right); // assume the longest
 // but the answer not limited to path the root
 return max(res, max(diameterOfBinaryTree(root->left), diameterOfBinaryTree(
 })
 int getDeep(TreeNode* node){
 if(!node) return 0;
 if(deepOfNode.count(node)) return deepOfNode[node];
 int h = 1 + max(getDeep(node->left), getDeep(node->right));
 return deepOfNode[node] = h;
 }
private:
 unordered_map<TreeNode*, int> deepOfNode; // use hash map to record the deep of
};

```

將兩個Recursive合併成一個，計算深度時順便存下最深

```

/*
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 int diameterOfBinaryTree(TreeNode* root) {
 int res=0;
 getDepth(root, res);
 return res;
 }
 int getDepth(TreeNode* node, int& res){
 if(!node) return 0;
 int left = getDepth(node->left, res);
 int right = getDepth(node->right, res);
 res = max(res, left+right); // record the Max depth
 return 1 + max(left, right); // count the Depth
 }
};

```

利用hash map優化這個方法

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 unordered_map<TreeNode*, int> deepOfnode;
 int diameterOfBinaryTree(TreeNode* root) {
 int res=0;
 getDepth(root, res);
 return res;
 }
 int getDepth(TreeNode* node, int& res){
 if(!node) return 0;
 if (deepOfnode.count(node)) return deepOfnode[node];
 int left = getDepth(node->left, res);
 int right = getDepth(node->right, res);
 res = max(res, left+right); // record the Max depth
 return deepOfnode[node] = 1 + max(left, right); // count the Depth
 }
};

```

## \*687. Longest Univalue Path | 5/1

---

Given a binary tree, find the length of the longest path where each node in the path has the same value. This path may or may not pass through the root.

The length of path between two nodes is represented by the number of edges between them.

Example 1:

Input:

```
 5
 / \
 4 5
 / \ \
 1 1 5
```

Output:

2

Example 2:

Input:

```
 1
 / \
 4 5
 / \ \
 4 4 5
```

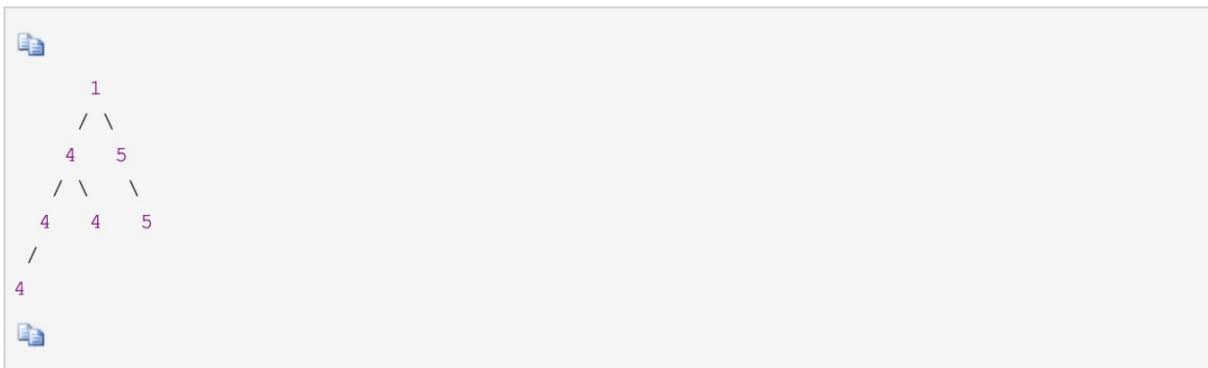
Output:

2

Note: The given binary tree has not more than 10000 nodes. The height of the tree is not more than 1000.

## 思路

解法1的思路：



若此時的node是只有兩個結點的第二層的那個結點4，那麼分別對其左右子結點調用遞歸，會得到  $left = 1$ ,  $right = 0$ ，因為此時要跟結點4組成path，所以肯定挑左子結點（有兩個4的那條邊），那您會問為啥不能連上右子結點的那個4，這整條長度為3的path ( $left+right$ , 此時的left和right已經分別自增1了,  $left=2$ ,  $right=1$ ) 實際我們已經用來更新過結果res了。需要注意的是我們的遞歸函數helper返回值的意義，並不是經過某個結點的最長路徑的長度，最長路徑長度保存在了結果res中，不是返回值，返回的是以該結點為終點的最長路徑長度，這樣回溯的時候，我們還可以繼續連上其父結點，比如若根結點也是4的話，那麼回溯到根結點的時候，路徑長度又可以增加了，參見代碼如下：

解法2思路：

下面這種解法跟上面的方法很類似，區別在於遞歸函數中多了一個參數，parent保存的是父結點值，其實仔細比較下兩種解法，發現就是加1的地方略有不同，那麼這裡helper的返回值意義就發生了變化，這裡的返回值表示的是以當前結點的父結點為路徑終點的最大相同值路徑長度，這樣我們遞歸出來的left和right就不用再加1，直接可以求和並更新結果res了，由於當前結點的父結點值知道，那麼我們和父結點值比較一下，如果相同，返回left和right中較大值並再加1，如果不同，則返回0，這是因為之前說的必須要以父結點為路徑終點，那麼既然父結點的值不同，所以長度肯定是0了，參見代碼如下：

## Code

這個方法與上題的方法相似，是由父親節點往下看

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 int longestUnivaluePath(TreeNode* root) {
 int res=0;
 getDepth(root, res); // use this func to "update" the max depth of each node
 return res;
 }
 int getDepth(TreeNode* node, int& res){
 if (!node) return 0;
 int left = getDepth(node->left, res);
 int right = getDepth(node->right, res);
 left = (node->left && node->left->val == node->val) ? left+1:0;
 right = (node->right && node->right->val == node->val) ? right+1:0;
 res = max(res, left+right); // the longest path is the node's left + right
 return max(left, right);
 }
};
```

而這個方法是由子結點往父親節點看，回傳值包含父親節點

```
class Solution {
public:
 int longestUnivaluePath(TreeNode* root) {
 int res = 0;
```

```
 getDepth(root, res, root->val);
 return res;
 }
 int getDepth(TreeNode* node, int& res, int parentVal){
 if (!node) return 0;
 int left = getDepth(node->left, res, node->val); // 其左邊孩子如果包含自己可以多+1
 int right = getDepth(node->right, res, node->val);
 res = max(res, left+right);
 if (node->val == parentVal)
 return max(left, right)+1; //Note: we return the value which include the parentVal
 else
 return 0;
 }
};
```

## \*124. Binary Tree Maximum Path Sum | 5/1

Given a non-empty binary tree, find the maximum path sum.

For this problem, a path is defined as any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The path must contain at least one node and does not need to go through the root.

**Example 1:**

**Input:** [1,2,3]



**Output:** 6

**Example 2:**

**Input:** [-10,9,20,null,null,15,7]



**Output:** 42

## 5/6review

在宣告result時要宣告成INT\_MIN來防止[-3] 這種testcase

## 思路

```
4
/ \
11 13
/ \
7 2
```

由於這是一個很簡單的例子，我們很容易就能找到最長路徑為7-11-4-13，那麼怎麼用遞歸來找出正確的路徑和呢？根據以往的經驗，樹的遞歸解法一般都是遞歸到葉節點，然後開始邊處理邊回溯到根節點。那麼我們就假設此時已經遞歸到結點7了，那麼其沒有左右子節點，所以如果以結點7為根結點的子樹最大路徑和就是7。然後回溯到結點11，如果以結點11為根結點的子樹，我們知道最大路徑和為 $7+11+2=20$ 。但是當回溯到結點4的時候，對於結點11來說，就不能同時取兩條路徑了，只能取左路徑，或者是右路徑，所以當根結點是4的時候，那麼結點11只能取其左子結點7，因為7大於2。所以，對於每個結點來說，我們要知道經過其左子結點的path之和還是經過右子節點的path之和大。那麼我們的遞歸函數返回值就可以定義為以當前結點為根結點，到葉節點的最大路徑之和，然後全局路徑最大值放在參數中，用結果res來表示。

在遞歸函數中，如果當前結點不存在，那麼直接返回0。否則就分別對其左右子節點調用遞歸函數，由於路徑和有可能為負數，而我們當然不希望加上負的路徑和，所以我們和0相比，取較大的那個，就是要麼不加，加就要加正數。然後我們來更新全局最大值結果res，就是以左子結點為終點的最大path之和加上以右子節點為終點的最大path之和，還要加上當前結點值，這樣就組成了一個條完整的路徑。而我們返回值是取left和right中的較大值加上當前結點值，因為我們返回值的定義是以當前結點為終點的path之和，所以只能取left和right中較大的那個值，而不是兩個都要，參見代碼如下：

## Code

```
/**
 * Definition for a binary tree node
 */
struct TreeNode {
 int val;
 TreeNode *left;
 TreeNode *right;
 TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

class Solution {
public:
 int maxPathSum(TreeNode* root) {
 int res = INT_MIN;
 findMaxPath(root, res);
 return res;
 }

 int findMaxPath(TreeNode* node, int& res){
 if (!node) return 0;
 int left = max(findMaxPath(node->left, res), 0); //如果左邊的值有比0大，才加
 int right = max(findMaxPath(node->right, res), 0);

 res = max(res, left+right+ node->val);
 return max(left, right) + node->val;
 }
};
```



## @129. Sum Root to Leaf Numbers | 5/3

Given a binary tree containing digits from 0-9 only, each root-to-leaf path could represent a number.

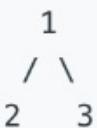
An example is the root-to-leaf path 1->2->3 which represents the number 123.

Find the total sum of all root-to-leaf numbers.

Note: A leaf is a node with no children.

### Example:

**Input:** [1,2,3]



**Output:** 25

#### Explanation:

The root-to-leaf path 1->2 represents the number 12.

The root-to-leaf path 1->3 represents the number 13.

Therefore, sum = 12 + 13 = 25.

### Example 2:

**Input:** [4,9,0,5,1]



**Output:** 1026

#### Explanation:

The root-to-leaf path 4->9->5 represents the number 495.

The root-to-leaf path 4->9->1 represents the number 491.

The root-to-leaf path 4->0 represents the number 40.

Therefore, sum = 495 + 491 + 40 = 1026.

# 思路

這道求根到葉節點數字之和的題跟之前的求 Path Sum 很類似，都是利用DFS遞歸來解，這道題由於不是單純的把各個節點的數字相加，而是每遇到一個新的子節點的數字，要把父節點的數字擴大10倍之後再相加。

## Code

Recursive:

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 int sumNumbers(TreeNode* root) {
 int sum=0;
 sum = sumOfroot(root, sum);
 return sum;
 }

 int sumOfroot(TreeNode* p, int sum){
 if (!p) return 0;
 sum = sum * 10 + p->val;
 if (!p->left && !p->right) return sum;
 else return sumOfroot(p->left, sum) + sumOfroot(p->right, sum);
 }
};
```

Iterative:

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
```

```

public:
 int sumNumbers(TreeNode* root) {
 int sum=0;
 stack<TreeNode*> stack;
 stack.push(root);
 while(!stack.empty()){
 TreeNode* p = stack.top(); stack.pop();
 if (!p->left && !p->right)
 sum += p->val;
 if (p->right){
 p->right->val += p->val*10;
 stack.push(p->right);
 }
 if (p->left){
 p->left->val += p->val* 10;
 stack.push(p->left);
 }
 }
 return sum;
 }
};

```

## @257. Binary Tree Paths | 5/3

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

Note: A leaf is a node with no children.

### Example:

#### Input:



**Output:** ["1->2->5", "1->3"]

**Explanation:** All root-to-leaf paths are: 1->2->5, 1->3

FB interview: recursive way may cause overflow when tree is super large.

Can you do it with iterative way?

## 思路

在遞歸函數中，當我們遇到葉結點的時候，即沒有左右子結點，那麼此時一條完整的路徑已經形成了，我們加上當前的葉結點後存入結果res中，然後回溯。注意這裡結果res需要reference，而out是不需要引用的，不然回溯回去還要刪除新添加的結點，很麻煩。為了減少判斷空結點的步驟，我們在調用遞歸函數之前都檢驗一下非空即可

## Code

Recursive:

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 vector<string> binaryTreePaths(TreeNode* root) {
 if (!root) return {};
 vector<string> res;
 dfs(root, res, "");
 return res;
 }
 void dfs(TreeNode* p, vector<string>& res, string out){
 if (!p->left && !p->right){
 res.push_back(out + to_string(p->val));
 return;
 }
 if (p->left) dfs(p->left, res, out + to_string(p->val) + "->");
 if (p->right) dfs(p->right, res, out + to_string(p->val) + "->");
 }
};
```

Iterative:

```
class Solution {
public:
 vector<string> binaryTreePaths(TreeNode* root) {
 vector<string> res;
 if (!root) return res;
 stack<TreeNode*> s;
 stack<string> ss;
```

```

s.push(root);
ss.push(to_string(root->val));

while (!s.empty()){
 TreeNode* cur = s.top(); s.pop();
 string str = ss.top(); ss.pop();

 if (!cur->left && !cur->right){
 res.push_back(str);
 continue;
 }
 if (cur->left){
 s.push(cur->left);
 ss.push(str + "->" + to_string(cur->left->val));
 }
 if (cur->right){
 s.push(cur->right);
 ss.push(str + "->" + to_string(cur->right->val));
 }
}
return res;
};

};

```

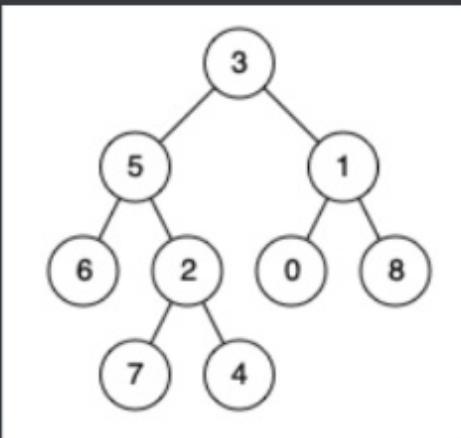
## 236. Lowest Common Ancestor of a Binary Tree | 5/28

---

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

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

Given the following binary tree: root = [3,5,1,6,2,0,8,null,null,7,4]



Example 1:

**Input:** root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1

**Output:** 3

**Explanation:** The LCA of nodes 5 and 1 is 3.

Example 2:

**Input:** root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4

**Output:** 5

**Explanation:** The LCA of nodes 5 and 4 is 5, since a node can be a descendant of itself according to the LCA definition.

Note:

- All of the nodes' values will be unique.
- p and q are different and both values will exist in the binary tree.

## 思路

由題可知，p和q只有三種情況

1. p q 分別在左和右，那此輪recursive的root便為解
2. p q 都在左，那我們便只需要找出p和q最高的位置，也就是遞迴第一個碰到p or q的root，此時left便等於該root，且right必為空

3. p q 都在右，(同上)，right便等於該root，且left必為空

## Code

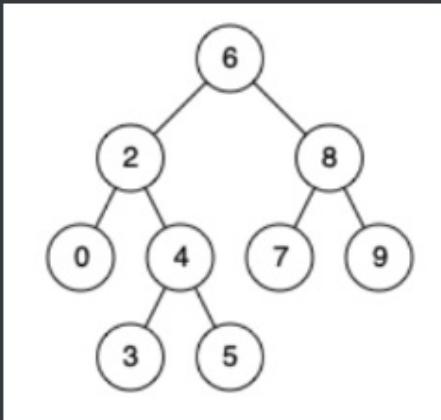
```
class Solution {
public:
 TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
 if (!root || root == p || root == q) return root; // search
 TreeNode* left = lowestCommonAncestor(root->left, p, q);
 TreeNode* right = lowestCommonAncestor(root->right, p, q);
 if (left && right) return root;
 return left ? left:right; // if left exist, return left or otherwise
 }
};
```

## 235. Lowest Common Ancestor of a Binary Search Tree | 5/28

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 p and q as the lowest node in T that has both p and q as descendants (where we allow a node to be a descendant of itself).”

Given binary search tree: root = [6,2,8,0,4,7,9,null,null,3,5]



**Example 1:**

**Input:** root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8

**Output:** 6

**Explanation:** The LCA of nodes 2 and 8 is 6.

**Example 2:**

**Input:** root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4

**Output:** 2

**Explanation:** The LCA of nodes 2 and 4 is 2, since a node can be a descendant of itself according to the LCA definition.

**Note:**

- All of the nodes' values will be unique.
- p and q are different and both values will exist in the BST.

## 思路

因為是二元搜尋樹，因此判斷 p q 條件容易許多  
若中間節點大於p跟q，表示pq均落在左子樹  
若中間節點小於p跟q，表示pq均落在右子數  
反之 p q 分佈於左右兩邊，則此種情況，中間節點即為解

## Code

```
class Solution {
public:
 TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
 if (!root) return NULL;

 // if the root val larger than p or q, means that p and q both in the left
 if (root->val > max(p->val, q->val))
 return lowestCommonAncestor(root->left, p, q);

 // Otherwise
 if (root->val < min(p->val, q->val))
 return lowestCommonAncestor(root->right, p, q);

 return root;
 }
};
```

## 297. Serialize and Deserialize Binary Tree | 5/28

---

Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment.

Design an algorithm to serialize and deserialize a binary tree. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary tree can be serialized to a string and this string can be deserialized to the original tree structure.

## Example:

You may serialize the following tree:



as "[1,2,3,null,null,4,5]"

**Clarification:** The above format is the same as how LeetCode serializes a binary tree. You do not necessarily need to follow this format, so please be creative and come up with different approaches yourself.

**Note:** Do not use class member/global/static variables to store states. Your serialize and deserialize algorithms should be stateless.

## 思路

先序遍歷的遞歸解法，非常的簡單易懂，我們需要接入輸入和輸出字符串流`istringstream`和`ostringstream`，對於序列化，我們從根節點開始，如果節點存在，則將值存入輸出字符串流，然後分別對其左右子節點遞歸調用序列化函數即可。對於去序列化，我們先讀入第一個字符，以此生成一個根節點，然後再對根節點的左右子節點遞歸調用去序列化函數即可

## Code

```
class Codec {
public:
```

```

// Encodes a tree to a single string.
string serialize(TreeNode* root) {
 ostringstream out;
 serialize(root, out);
 return out.str();
}

// Decodes your encoded data to tree.
TreeNode* deserialize(string data) {
 istringstream in(data);
 return deserialize(in);
}

private:
 void serialize(TreeNode* root, ostringstream& out){
 if (!root) out << "# ";
 else{
 out << root->val << " ";
 serialize(root->left, out);
 serialize(root->right, out);
 }
 //return out.str(); // call by refrence will reduce time and space
 }

 TreeNode* deserialize(istringstream& in){
 string val;
 in >> val;
 if (val == "#") return nullptr;
 TreeNode* root = new TreeNode(stoi(val));
 // istringstream will automatically move forward
 root->left = deserialize(in);
 root->right = deserialize(in);
 return root;
 }
};

// Your Codec object will be instantiated and called as such:
// Codec codec;
// codec.deserialize(codec.serialize(root));

```

## 508. Most Frequent Subtree Sum | 5/28

---

Given the root of a tree, you are asked to find the most frequent subtree sum. The subtree sum of a node is defined as the sum of all the node values formed by the subtree rooted at that node (including the node itself). So what is the most frequent subtree sum value? If there is a tie, return all the values with the highest frequency in any order.

## Examples 1

Input:



return [2, -3, 4], since all the values happen only once, return all of them in any order.

## Examples 2

Input:



return [2], since 2 happens twice, however -5 only occurs once.

**Note:** You may assume the sum of values in any subtree is in the range of 32-bit signed integer.

## 思路

根據subsum tree的定義可以發現，這棵樹由底下往上建可以減少重複運算的機會  
因此在走訪這棵樹時我們採取post order。

函數回傳sum值，並recursive往上建樹，同時更新sum值出現頻率的map。

最後回到main function再根據頻率去更新res，即可求出答案。

(max count的計算可在函數中完成或者等到回main再求出)

## Code

```

class Solution {
public:
 vector<int> findFrequentTreeSum(TreeNode* root) {
 vector<int> res;
 map<int, int> m; // sum to cnt
 int cnt = 0;
 postorder(root, m, cnt);
 for (pair<int, int> i : m){
 if (i.second == cnt){
 res.push_back(i.first);
 }
 }
 return res;
 }
 int postorder(TreeNode* root, map<int, int>& m, int& cnt){
 if (!root) return 0;
 int left = postorder(root->left, m, cnt);
 int right = postorder(root->right, m, cnt);
 int sum = left + right + root->val;
 cnt = max(cnt, ++m[sum]);
 return sum;
 }
};

```

優化：在跑完post order後再來將max cnt求出，減少function內的參數

```

class Solution {
public:
 vector<int> findFrequentTreeSum(TreeNode* root) {
 vector<int> res;
 map<int, int> m; // sum to cnt
 //int cnt = 0;
 postorder(root, m);
 int cnt = -1;
 for (pair<int, int> i : m){ // get the max count
 if (i.second > cnt) cnt = i.second;
 }
 //postorder(root, m, cnt);
 for (pair<int, int> i : m){
 if (i.second == cnt){
 res.push_back(i.first);
 }
 }
 return res;
 }
 int postorder(TreeNode* root, map<int,int>& m){
//int postorder(TreeNode* root, map<int, int>& m, int& cnt){
 if (!root) return 0;
 int left = postorder(root->left, m);
 int right = postorder(root->right, m);
 int sum = left + right + root->val;

```

```
//cnt = max(cnt, ++m[sum]);
m[sum]++;
return sum;
}
};
```

---

## 337. House Robber III | 5/29

---

The thief has found himself a new place for his thievery again. There is only one entrance to this area, called the "root." Besides the root, each house has one and only one parent house. After a tour, the smart thief realized that "all houses in this place forms a binary tree". It will automatically contact the police if two directly-linked houses were broken into on the same night.

Determine the maximum amount of money the thief can rob tonight without alerting the police.

### Example 1:

**Input:** [3,2,3,null,3,null,1]



**Output:** 7

**Explanation:** Maximum amount of money the thief can rob =  $3 + 3 + 1 = 7$ .

### Example 2:

**Input:** [3,4,5,1,3,null,1]



**Output:** 9

**Explanation:** Maximum amount of money the thief can rob =  $4 + 5 = 9$ .

## 思路

這題要求的其實本質上是sum of subtree，  
只是這裡的sum有點限制。

這裡的限制所帶出來的概念就是，  
選擇此root，或者選擇left or right。  
這個概念可以運用到所有節點上，如何選擇才會讓這個sum最大。  
因此比較的點落在：

1. root + left的left + left的right + right的left + right的right

v.s.

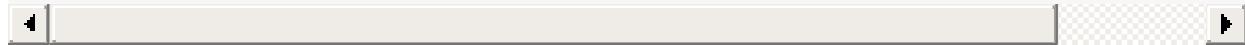
2. left + right

並且在遞迴過程中，都會用一個hash map記錄怎麼選擇對於該節點會是最大的結果  
並且讓樹由下往上生成，即可求解。

## Code

```
class Solution {
public:
 int rob(TreeNode* root) {
 unordered_map<TreeNode*, int> cache; // cache of max_val of each node as root
 return getMax(root, cache);
 }

 int getMax(TreeNode* node, unordered_map<TreeNode*, int>& cache){
 if (!node) return 0;
 if (cache.count(node)) return cache[node];
 int val = 0;
 if (node->left){
 val += getMax(node->left->left, cache) + getMax(node->left->right, cache);
 }
 if (node->right){
 val += getMax(node->right->left, cache) + getMax(node->right->right, cache);
 }
 val = max(val+node->val , getMax(node->left, cache) + getMax(node->right, cache));
 cache[node] = val;
 return val;
 }
};
```



## 98. Validate Binary Search Tree | 5/30

Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

- The left subtree of a node contains only nodes with keys less than the node's key.

- The right subtree of a node contains only nodes with keys greater than the node's key.
- Both the left and right subtrees must also be binary search trees.

### Example 1:

```
2
/ \
1 3
```

**Input:** [2,1,3]

**Output:** true

### Example 2:

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

**Input:** [5,1,4,null,null,3,6]

**Output:** false

**Explanation:** The root node's value is 5 but its right child's value is 4.

## 思路

這道驗證二叉搜索樹有很多種解法，可以利用它本身的性質來做，即左<根<右，也可以通過利用中序遍歷結果為有序數列來做，下面我們先來看最簡單的一種，就是利用其本身性質來做，初始化時帶入系統最大值和最小值，在遞歸過程中換成它們自己的節點值，用long代替int就是為了包括int的邊界條件

## Code

runtime: 8ms

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 bool isValidBST(TreeNode* root) {
 return validation(root, LONG_MIN, LONG_MAX);
 }
 bool validation(TreeNode* root, long min, long max){
 if (!root) return true;
 else if (root->val <= min || root->val >= max) return false; // should larger
 else return validation(root->left, min, root->val) && validation(root->right, root->val);
 }
};

```



24 ms

runtime 差三倍只差在有沒有用else if 跟 else

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 * int val;
 * TreeNode *left;
 * TreeNode *right;
 * TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
 bool isValidBST(TreeNode* root) {
 return validation(root, LONG_MIN, LONG_MAX);
 }
 bool validation(TreeNode* root, long min, long max){
 if (!root) return true;
 if (root->val <= min || root->val >= max) return false; // should larger
 return validation(root->left, min, root->val) && validation(root->right, root->val);
 }
};

```



## 530. Minimum Absolute Difference in BST | 5/30

Given a binary search tree with non-negative values, find the minimum absolute difference between values of any two nodes.

**Example:**

**Input:**

```
1
 \
 3
 /
2
```

**Output:**

```
1
```

**Explanation:**

The minimum absolute difference is 1, which is the difference between 2 and 1 (or between 2 and 3).

**Note:** There are at least two nodes in this BST.

### 思路

這道題給了我們一棵二叉搜索樹，讓我們求任意個節點值之間的最小絕對差。由於BST的左<根<右的性質可知，如果按照中序遍歷會得到一個有序數組，那麼最小絕對差肯定在相鄰的兩個節點值之間產生。所以我們的做法就是對BST進行中序遍歷，然後當前節點值和之前節點值求絕對差並更新結果res。這裡需要注意的就是在處理第一個節點值時，由於其沒有前節點，所以不能求絕對差。這裡我們用變量pre來表示前節點值，這裡由於題目中說明了所以節點值不為負數，所以我們給pre初始化-1，這樣我們就知道pre是否存在。如果沒有題目中的這個非負條件，那麼就不能用int變量來，必須要用指針，通過來判斷是否為指向空來判斷前結點是否存在。還好這裡簡

化了問題，用-1就能搞定了

## Code

```
class Solution {
public:
 int getMinimumDifference(TreeNode* root) {
 int res = INT_MAX, pre = -1;
 inorder(root, pre, res);
 return res;
 }
 void inorder(TreeNode* root, int& pre, int& res){
 if (!root) return;
 inorder(root->left, pre, res);
 if (pre != -1) res = min(res, root->val-pre); // there are no pre in the f
 pre = root->val;
 inorder(root->right, pre, res);
 }
};
```



## 700. Search in a Binary Search Tree | 6/3

Given the root node of a binary search tree (BST) and a value. You need to find the node in the BST that the node's value equals the given value. Return the subtree rooted with that node. If such node doesn't exist, you should return NULL.

For example,

Given the tree:



And the value to search: 2

You should return this subtree:



In the example above, if we want to search the value 5, since there is no node with value 5, we should return NULL.

Note that an empty tree is represented by NULL, therefore you would see the expected output (serialized tree format) as [], not null.

## 思路

BST 性質

這就是一個天然的二分麼，當仁不讓的二分搜索法呼之慾出啊～

## Code

```
class Solution {
public:
 TreeNode* searchBST(TreeNode* root, int val) {
```

```
 if (!root) return NULL;
 if (root->val == val) return root;
 return val >= root->val ? searchBST(root->right, val) : searchBST(root->left,
);
};
```

## 701. Insert into a Binary Search Tree | 6/3

---

Given the root node of a binary search tree (BST) and a value to be inserted into the tree, insert the value into the BST. Return the root node of the BST after the insertion. It is guaranteed that the new value does not exist in the original BST.

Note that there may exist multiple valid ways for the insertion, as long as the tree remains a BST after insertion. You can return any of them.

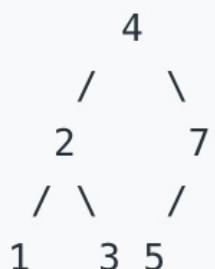
For example,

Given the tree:

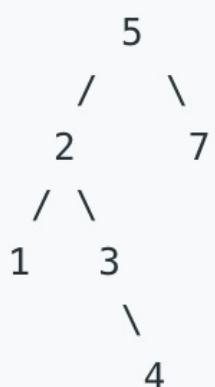


And the value to insert: 5

You can return this binary search tree:



This tree is also valid:



## 思路

這道題讓我們在二叉搜索樹中插入結點，當前還需要保持二叉搜索樹的性質，那麼插入結點的方

式就有多種，就像題目中給的那個例子。結點5可以有不同的方法，但是很顯然，放在結點7的左子結點比代替結點4成為根結點要來的簡單許多。怎麼簡單我們就怎麼來，所以還是按照簡單的來吧。由於二叉搜索樹自帶二分的性質，那麼首先根結點比較，如果大於根結點值的話，說明肯定要插入到右子樹中。所以接下來跟7比較，對於遞歸函數來說，結點7也可以當作是一個新的根結點，那麼由於結點7的值大於目標值5，所以要去其左子樹，我們發現其左子結點為空，那麼我們就可以根據目標值來生成一個新的結點，然後連到結點7的左子樹上即可。那麼在遞歸函數中，首先判斷當前結點是否為空，為空的話就新建一個結點返回。否則就判斷當前結點值是否大於目標值，是的話就對左子結點調用遞歸函數，並將返回值賦給當前結點的左子結點，否則就對右子結點調用遞歸函數，並將返回值賦給當前結點的右子結點，最後返回當前結點即可

## Code

Recursive:

```
class Solution {
public:
 TreeNode* insertIntoBST(TreeNode* root, int val) {
 if (!root) return new TreeNode(val);
 if (root->val > val) root->left = insertIntoBST(root->left, val); // should
 if (root->val < val) root->right = insertIntoBST(root->right, val);
 return root;
 }
};
```

Iterative:

```
class Solution {
public:
 TreeNode* insertIntoBST(TreeNode* root, int val) {
 if (!root) return NULL;
 TreeNode* cur = root;
 while(true){
 if (val > cur->val){
 if (!cur->right){
 cur->right = new TreeNode(val);
 break;
 }
 else cur = cur->right;
 }
 else{
 if (!cur->left){
 cur->left = new TreeNode(val);
 break;
 }
 else cur = cur->left;
 }
 }
 }
};
```

```
 }
 }
 return root;
}
};
```

---

## 230. Kth Smallest Element in a BST | 6/3

---

Given a binary search tree, write a function `kthSmallest` to find the  $k$ th smallest element in it.

Note:

You may assume  $k$  is always valid,  $1 \leq k \leq \text{BST's total elements}$ .

### Example 1:

**Input:** root = [3,1,4,null,2], k = 1



**Output:** 1

### Example 2:

**Input:** root = [5,3,6,2,4,null,null,1], k = 3



**Output:** 3

### Follow up:

What if the BST is modified (insert/delete operations) often and you need to find the kth smallest frequently? How would you optimize the kthSmallest routine?

## 思路

search in BST -> INORDER

both Recursive use counting way to find the answer

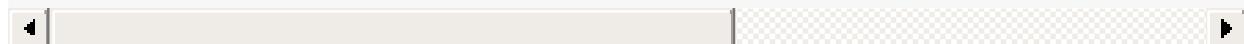
1. Recursive use counting-down way. Counting til zero, then returning that answer back to every state.

2. Iterative use counting-up way. Counting til k, then break the while-loop.
3. Divide and Conquer way is the optimal solution. If the  $k \leq$  the number of node in left tree means ans is on tree. if  $k >$  the number of node in the left tree means ans is on the right tree. if  $k = \text{count} + 1$  means the root of this state is the ans.
4. Follow UP: 這道題的Follow up中說假設該BST被修改的很頻繁，而且查找第k小元素的操作也很頻繁，問我們如何優化。其實最好的方法還是像上面的解法那樣利用分治法來快速定位目標所在的位置，但是每個遞歸都遍歷左子樹所有結點來計算個數的操作並不高效，所以我們應該修改原樹結點的結構，使其保存包括當前結點和其左右子樹所有結點的個數，這樣我們使用的時候就可以快速得到任何左子樹結點總數來幫我們快速定位目標值了。

## Code

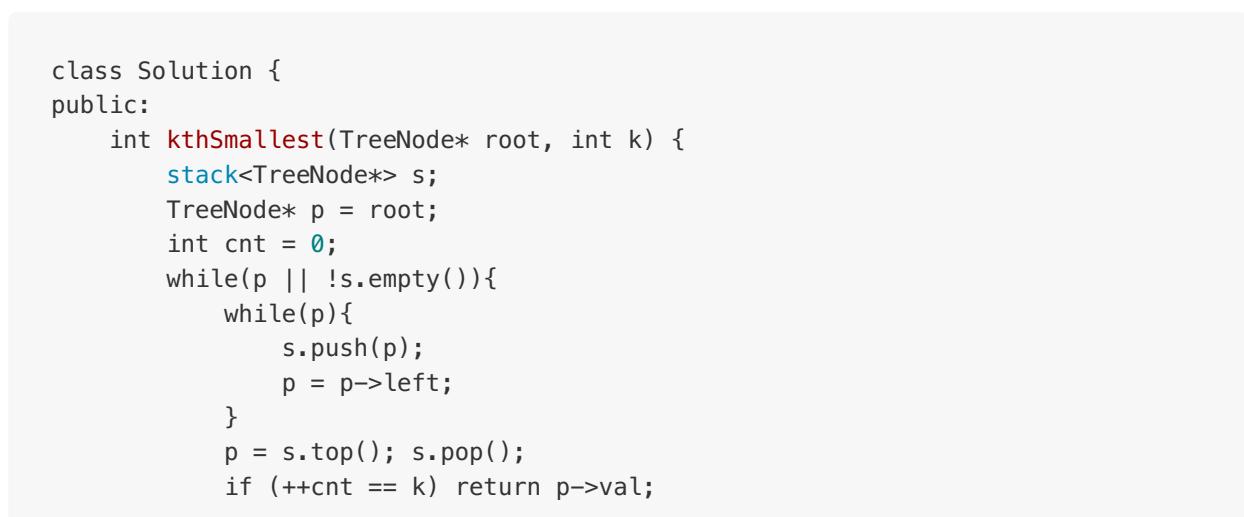
Recursive:

```
class Solution {
public:
 int kthSmallest(TreeNode* root, int k) {
 return kthSmallestDFS(root, k);
 }
 int kthSmallestDFS(TreeNode* root, int& k){
 if (!root) return -1;
 int val = kthSmallestDFS(root->left, k);
 if (k == 0) // have already find it from last state, so store at val, and r
 return val;
 if (--k == 0) // this state find the answer, so return the last state
 return root->val;
 return kthSmallestDFS(root->right, k);
 }
};
```



Iterative:

```
class Solution {
public:
 int kthSmallest(TreeNode* root, int k) {
 stack<TreeNode*> s;
 TreeNode* p = root;
 int cnt = 0;
 while(p || !s.empty()){
 while(p){
 s.push(p);
 p = p->left;
 }
 p = s.top(); s.pop();
 if (++cnt == k) return p->val;
 }
 }
};
```



```

 p = p->right;
 }
 return -1;
}

};

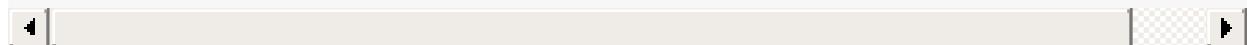
```

Divide&Conquer:

```

class Solution {
public:
 int kthSmallest(TreeNode* root, int k) {
 int cnt = countnode(root->left); // count the number of the node in left tree
 if (k <= cnt){ // means the target is on the left tree
 return kthSmallest(root->left, k);
 }
 else if (k > cnt+1){ // means the target is on the right tree
 return kthSmallest(root->right, k - cnt - 1);
 }
 else return root->val; // k = cnt + 1 is the answer
 }
 int countnode(TreeNode* t){
 if (!t) return 0;
 else return 1 + countnode(t->left) + countnode(t->right);
 }
};

```



Follow up:

What if the BST is modified (insert/delete operations) often and you need to find the kth smallest frequently?

How would you optimize the kthSmallest routine?

```

class Solution {
public:
 struct MyTreeNode{
 int val;
 int count; // counting the total of child-nodes this node have
 MyTreeNode* left;
 MyTreeNode* right;
 MyTreeNode(int x): val(x), count(1), left(NULL), right(NULL){}
 };

 MyTreeNode* build(TreeNode* root){
 if (!root) return NULL;
 MyTreeNode* node = new MyTreeNode(root->val);
 node->left = build(root->left);

```

```

 node->right = build(root->right);
 if (root->left) node->count += node->left->count;
 if (root->right) node->count += node->right->count;
 return node;
 }

 int kthSmallest(TreeNode* root, int k) {
 MyTreeNode* node = build(root); // updating node->count
 return dfs(node, k);
 }
 int dfs(MyTreeNode* node, int k){
 // step1: locate the ans -> left tree node counts
 if (node->left){
 int cnt = node->left->count;
 if (k <= cnt) {
 return dfs(node->left, k);
 }
 else if (k > cnt + 1){
 return dfs(node->right, k-cnt-1);
 }
 else return node->val; // k = cnt + 1
 }
 else{
 if (k == 1) return node->val;
 else return dfs(node->right, k-1);
 }
 }
};

```

## 99. Recover Binary Search Tree | 6/13

Two elements of a binary search tree (BST) are swapped by mistake.

Recover the tree without changing its structure.

### Example 1:

**Input:** [1,3,null,null,2]

```

 1
 /
 3
 \
 2

```

**Output:** [3,1,null,null,2]

```
3
/
1
 \
 2
```

### Example 2:

**Input:** [3,1,4,null,null,2]

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

**Output:** [2,1,4,null,null,3]

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

### Follow up:

- A solution using  $O(n)$  space is pretty straight forward.
- Could you devise a constant space solution?

## 思路

We aren't actually swapping the node, what contains in nodes is not only the val but also the link,

the left and right pointer. At this point, we don't want to break the structure of the tree

by swapping the nodes directly. Instead, we only swap the value. Specifically, we need to list to do so.

First is the list which can memorize the tree's structure. And the second is the list to memorize the value of the tree's order. Just before the end of the program, we sort the value list and assign

to the First list. And the first list is the final answer.

## Code

```
class Solution {
public:
 void recoverTree(TreeNode* root) {
 vector<TreeNode*> tree;
 vector<int> vals;
 inorder(root, tree, vals);
 sort(vals.begin(), vals.end());
 for (int i=0; i<vals.size(); i++){
 tree[i]->val = vals[i];
 }
 }
 void inorder(TreeNode* root, vector<TreeNode*>& tree, vector<int>& vals){
 if(!root) return;
 inorder(root->left, tree, vals);
 tree.push_back(root);
 vals.push_back(root->val);
 inorder(root->right, tree, vals);
 }
};
```

---

## 108. Convert Sorted Array to Binary Search Tree | 6/12

---

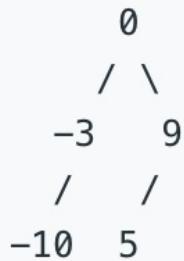
Given an array where elements are sorted in ascending order, convert it to a height balanced BST.

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.

## Example:

Given the sorted array: [-10,-3,0,5,9],

One possible answer is: [0,-3,9,-10,null,5], which represents the following height balanced BST:



## 思路

Based on the inorder of BST, we can get a sequential numbers.

So if we pick the center of a sequential number, we can get a height-balanced tree.

And recursively use this strategy to the left tree and the right one.

## Code

Naive:

```
class Solution:
 def sortedArrayToBST(self, nums: List[int]) -> TreeNode:
 return self.recur(nums)

 def recur(self, leaves):
 if len(leaves) == 0:
 return None

 l, r = 0, len(leaves)-1
 mid = l + (r-l)//2
 root = TreeNode(leaves[mid])
 root.left = self.recur(leaves[:mid])
 root.right = self.recur(leaves[mid+1:])

 return root
```

Save Space and Increase Time:

```
class Solution:
 def sortedArrayToBST(self, nums: List[int]) -> TreeNode:
 def recur(l, r):
 if l > r:
 return None

 mid = l + (r-l)//2
 root = TreeNode(nums[mid])
 root.left = recur(l, mid-1)
 root.right = recur(mid+1, r)

 return root

 return recur(0, len(nums)-1)
```

```
class Solution {
public:
 TreeNode* sortedArrayToBST(vector<int>& nums) {
 return recursive(nums, 0, nums.size()-1);
 }
 TreeNode* recursive(vector<int>& nums, int left, int right){
 if (left > right) return NULL;
 int mid = left + (right - left) / 2;
 TreeNode* node = new TreeNode(nums[mid]);
 node->left = recursive(nums, left, mid-1);
 node->right = recursive(nums, mid+1, right);
 return node;
 }
};
```

## 501. Find Mode in Binary Search Tree | 6/12

Given a binary search tree (BST) with duplicates, find all the mode(s) (the most frequently occurred element) in the given BST.

Assume a BST is defined as follows:

The left subtree of a node contains only nodes with keys less than or equal to the node's key.  
The right subtree of a node contains only nodes with keys greater than or equal to the node's key.

Both the left and right subtrees must also be binary search trees.

For example:

Given BST [ 1, null, 2, 2 ],



return [ 2 ].

**Note:** If a tree has more than one mode, you can return them in any order.

**Follow up:** Could you do that without using any extra space?  
(Assume that the implicit stack space incurred due to recursion does not count).

## 思路

There are two solution.

First is use the hashmap to memorize the frequency of each number.

As for the follow up we should use the optimal of space solution.

We should give up the hashmap.

Thus, we should consider the attributes of the BST.

If we travel the tree with inorder, the number would be sequential.

Which means we can expect the same value number sequentially shows up, when we travel the BST.

Specificly, we should use a "pre" to memorize the last node, and comparing to the node in this state.

If same, than add 1 to the count, if not assign 1 to the count.

Meanwhile, we should also update the max count.

If larger, we clear the ans vector and push the new max into the ans vector.

If same large, we just push the new max into the ans vector.

## Code

```
class Solution {
public:
 vector<int> findMode(TreeNode* root) {
 vector<int> ans;
 int count = 1;
 int max = 0;
 TreeNode* pre = NULL;
 findmostfreq(root, ans, pre, count, max);
 return ans;
 }

 void findmostfreq(TreeNode* node, vector<int>& ans, TreeNode*& pre, int& count,
 if (!node) return;
 findmostfreq(node->left, ans, pre, count, max);
 if (pre){
 count = (node->val == pre->val) ? count+1 : 1;
 }
 if (count >= max){
 if (count > max) ans.clear();
 ans.push_back(node->val);
 max = count;
 }
 pre = node;
 findmostfreq(node->right, ans, pre, count, max);
 }
 };
};
```

## 450. Delete Node in a BST | 6/14

Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return the root node reference (possibly updated) of the BST.

Basically, the deletion can be divided into two stages:

Search for a node to remove.

If the node is found, delete the node.

Note: Time complexity should be O(height of tree).

### Example:

```
root = [5,3,6,2,4,null,7]
key = 3
```



Given key to delete is 3. So we find the node with value 3 and delete it.

One valid answer is [5,4,6,2,null,null,7], shown in the following BST.



Another valid answer is [5,2,6,null,4,null,7].



思路

steps of delete: 1.locate the object 2.replace the object with suitable node

1. locate the object: if-else statement with BST attrb

2. replace:

the rule of replace:

if the left and right of the object doesn't exsist -> assign the object to NULL and done

if left or right exist -> assign to the exist node

!! if left and right exist -> assign to the smallest node in the right tree !!

## Code

```
class Solution {
public:
 TreeNode* deleteNode(TreeNode* root, int key) {
 if (!root) return NULL;
 if (key > root->val){
 root->right = deleteNode(root->right, key);
 }
 else if (key < root->val){
 root->left = deleteNode(root->left, key);
 }
 else{ // if locate the key, then do some replace
 if (!root->left || !root->right){ // combine rule 1 and 2 in same if
 root = (root->left)? root->left:root->right; // if both doesn't exist
 }
 else {
 TreeNode* Rnode = root->right;
 while (Rnode->left) Rnode = Rnode->left; // locate the smallest
 root->val = Rnode->val;
 root->right = deleteNode(root->right, Rnode->val); // delete the sm
 }
 }
 return root;
 }
};
```

## [Start of the Binary Search]

## Conclusion

<https://www.cnblogs.com/grandyang/p/6854825.html>

Case of the Exception of the Range: Leetcode: 108, 34

- What is the sign of Binary Search question:
    - i. limitation of the time complexity  $O(\log(n))$
    - ii. given an (part of) sorted array
  - Properties of Binary search
    - i. if the counts of array elems is even, then we pick the lower of the center
- (?) This rule is questionable !!!, just pick up a way and try with simple data [], [1], [1,2] ... etc.  
actually it depends on how you set r:  
if you set r to len(nums), then you use while  $l < r$ . when  $l == r$ , the search space becomes empty.  
if you set r to len(nums)-1(inclusive), then you should use while  $l \leq r$ . when  $l > r$ , the search space becomes empty.  
because eventually you need the while loop to be terminated with an empty range.

## Conclusion Update

1. Always use right = len(nums) -1, because you may want to access arr[right]!
    - i. So you have to use left  $\leq$  right in termination.
  2. Always use  $<$  or  $\leq$ , and always put the arr[mid] in the left.
- 

## 704. Binary Search | 6/15

---

Given a sorted (in ascending order) integer array nums of n elements and a target value, write a function to search target in nums. If target exists, then return its index, otherwise return -1.

### Example 1:

**Input:** `nums` = `[-1,0,3,5,9,12]`, `target` = 9

**Output:** 4

**Explanation:** 9 exists in `nums` and its index is 4

### Example 2:

**Input:** `nums` = `[-1,0,3,5,9,12]`, `target` = 2

**Output:** -1

**Explanation:** 2 does not exist in `nums` so return -1

### Note:

1. You may assume that all elements in `nums` are unique.
2. `n` will be in the range `[1, 10000]`.
3. The value of each element in `nums` will be in the range `[-9999, 9999]`.

## 思路

最基礎的Binary Search

## Code

```
class Solution {
public:
 int search(vector<int>& nums, int target) {
 if (nums.empty()) return -1;
 int left=0, right=nums.size();
 while (right > left){
 int mid = left + (right-left)/2;
```

```
 if (nums[mid] == target) return mid;
 else if (nums[mid] > target){
 // means target is on the right of the mid
 right = mid;
 }
 else{
 left = mid + 1;
 }
}
// if (nums[left] != target) return -1;
// else return left;
return -1;
};
```

---

## 35. Search Insert Position | 6/14

---

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

### Example 1:

```
Input: [1,3,5,6], 5
Output: 2
```

### Example 2:

```
Input: [1,3,5,6], 2
Output: 1
```

### Example 3:

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

### Example 4:

```
Input: [1,3,5,6], 0
Output: 0
```

## 思路

We can use binary search.

Use while-loop is okay here no need the recursion

## Code

```
class Solution:
 def searchInsert(self, nums: List[int], target: int) -> int:
 left, right = 0, len(nums)-1

 while left <= right:
 mid = left + (right-left)//2
```

```

 if nums[mid] < target:
 left = mid + 1
 else:
 right = mid - 1

return left

```

```

class Solution {
public:
 int searchInsert(vector<int>& nums, int target) {
 if (target > nums.back()) return nums.size();
 else{
 int left = 0, right = nums.size();
 while (right > left){ // stop at right = left
 int mid = left + (right-left)/2;
 if (nums[mid] == target) return mid;
 else if (nums[mid] > target){
 right = mid;
 }
 else left = mid + 1;
 }
 return left;
 }
 }
};

```

---

## 34. Find First and Last Position of Element in Sorted Array | 6/15

---

Given an array of integers `nums` sorted in ascending order, find the starting and ending position of a given target value.

Your algorithm's runtime complexity must be in the order of  $O(\log n)$ .

If the target is not found in the array, return  $[-1, -1]$ .

### Example 1:

```
Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]
```

### Example 2:

```
Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]
```

## 二刷思路更新

其實不需要像下面思考的這麼困難

原本的做法：

先用binary search找到target的前一個節點 -> right

那right+1就是第一個等於target的點

注意：如果right+1有可能會越界，就這個需要檢查而已

在用另一個binary search找到大於target的節點 -> left

那left-1就是最後一個等於target的點

## 思路

This question asks us to find the head and the end of the sequential target  
We can solve this by two ways

1. Find the target, then goes both side til encounter the difference ( O(n) )
2. use 2 Binary search to find head and the end. ( O(log(n)) )  
We implement way2.

Step1: find the head or the begin of the range

1. if A[mid] < target -> i = mid+1 (the head is on the right of the mid)

2. if  $A[mid] > target \rightarrow j = mid - 1$  (the head is on the left of the mid)

3. if  $A[mid] = target \rightarrow j = mid$  (the head is on the left of OR at the mid)

Note: We can combine 2 and 3 to one equation

If  $A[mid] \geq target, j = mid$ ;

Surprisingly, 1 and 2\* are the only logic you need to put in loop while ( $i < j$ ).

When the while loop terminates ( $i == j$ ), the value of  $i/j$  is where the start of the range is.

Why?

No matter what the sequence originally is, as we narrow down the search range, eventually we will be at a situation where there are only two elements in the s Suppose our target is 5, then we have only 7 possible cases:

i j

case 1: [5 7] ( $A[i] = target < A[j]$ )  
case 2: [5 5] ( $A[i] = target = A[j]$ )  
case 3: [5 3] ( $A[i] = target > A[j] \Rightarrow$  this can't happen in while( $i < j$ ))  
**case 4: [3 5] ( $A[j] = target > A[i]$ )**  
case 5: [3 7] ( $A[i] < target < A[j]$ )  
case 6: [3 4] ( $A[i] < A[j] < target$ )  
case 7: [6 7] ( $target < A[i] < A[j]$ )

For case 1, 2, if we follow the above rule, since  $mid = i \Rightarrow A[mid] = target$  in  
For case 4, since  $A[mid] < target$ , then set  $i = mid + 1$ . The loop terminates and  
For all other cases, by the time the loop terminates,  $A[i]$  is not equal to 5. S

Step2: find the end of the range

4. if  $A[mid] < target \rightarrow i = mid + 1$  (the end is on the right of the mid)

5. if  $A[mid] > target \rightarrow j = mid - 1$  (the end is on the left of the mid)

6. if  $A[mid] = target \rightarrow i = mid$  (the end is on the right of OR at the mid)

Note: We can combine 1 and 3 to one equation

If  $A[mid] \leq target$ , then  $i = mid$ ; HOWEVER it will caused problem:

This case will cause problem [5 7], target = 5

Now  $A[mid] = 5$ , then according to rule 2, we set  $i = mid$ . This practically does nothing because  $i$  is already equal to  $mid$ . As a result, the search range is not moved at all!

The solution is by using a small trick:

instead of calculating  $mid$  as  $mid = i + (j - i) / 2$ , we now do:  $i + (j - i + 1) / 2$

When we use  $mid = i + (j - i) / 2$ , the  $mid$  is rounded to the lowest integer.

In other words, mid is always biased towards the left.

So in order to keep the search range moving, make sure the new i is set to something different than mid

Therefore,  $i + (j-i+1)/2$ , biased towards the right

## Code

Two binary search

```
class Solution:
 def searchRange(self, nums: List[int], target: int) -> List[int]:
 if len(nums) == 0:
 return [-1,-1]

 # find the start of the target
 # = find the one less than target then plus one. That is the start
 left, right = 0, len(nums)-1

 while left <= right:
 mid = left + (right-left)//2
 if nums[mid] < target:
 left = mid + 1
 else:
 right = mid - 1

 start = right+1
 if start >= len(nums) or nums[start] != target:
 return [-1,-1] # can't find

 # Use another binary search to find the first different int from the num[start]
 left, right = start, len(nums) -1

 while left <= right:
 mid = left + (right-left) //2
 if nums[mid] == target:
 left = mid + 1
 else:
 right = mid - 1
 return [start, left-1]
```

Binary search + Iterating find

```
class Solution:
 def searchRange(self, nums: List[int], target: int) -> List[int]:
```

```

if len(nums) == 0:
 return [-1,-1]

find the start of the target
= find the one less than target then plus one. That is the start
left, right = 0, len(nums)-1

while left <= right:
 mid = left + (right-left)//2
 if nums[mid] < target:
 left = mid + 1
 else:
 right = mid - 1

start = right+1
end = start
if start >= len(nums) or nums[start] != target:
 return [-1,-1] # can't find

iterating to find the position of the end of the target

for i in range(start+1, len(nums)):
 if nums[i] != nums[start]:
 break
 else:
 end = i

return [start, end]

```

```

class Solution {
public:
 vector<int> searchRange(vector<int>& nums, int target) {
 vector<int> res(2, -1);
 if (nums.empty()) return res;
 int left = 0, right = nums.size() - 1;
 // Search the head of the range
 while (left < right){
 int mid = left + (right-left)/2;
 if (nums[mid] < target){ // the head of range is on right
 left = mid + 1;
 }
 else{ // equation 2+3 -> nums[mid] >= mid
 right = mid;
 }
 }
 if (nums[left] != target) return res; // can't find any target
 else res[0] = left;

 // Search the end of the range
 right = nums.size() -1; // reset the right position
 while (left < right){
 int mid = left + (right-left+1)/2; // BIASED TOWARDS THE RIGHT to avoid

```

```
 if (nums[mid] > target){ // the end of range is on the left
 right = mid - 1;
 }
 else{ // equation 1+3 -> nums[mid] >= mid
 left = mid;
 }
 }
 res[1] = right;

 return res;
}
};
```

## 33. Search in Rotated Sorted Array | 6/17

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2]).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

Your algorithm's runtime complexity must be in the order of O(log n).

### Example 1:

```
Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4
```

### Example 2:

```
Input: nums = [4,5,6,7,0,1,2], target = 3
Output: -1
```

## 思路

Ascending order is the important rule in Binary Search,  
we have to find it out to enhance our efficient.

if we listed all the possibility of the rotation

0 1 2 4 5 6 7

7 0 1 2 4 5 6

6 7 0 1 2 4 5

5 6 7 0 1 2 4

4 5 6 7 0 1 2

2 4 5 6 7 0 1

1 2 4 5 6 7 0

0 1 2 4 5 6 7

7 0 1 2 4 5 6

6 7 0 1 2 4 5

5 6 7 0 1 2 4

4 5 6 7 0 1 2

2 4 5 6 7 0 1

1 2 4 5 6 7 0

we can find out:

if the middle of the array if smaller than the rightest element  
then the right of the array is ascending order; otherwise the left is ascending order.  
therefore we can find if the target is in the ascending area,  
combining these two conditions, we can finally said that  
to decide whether the target is in the left or the right side of the array.

## Code

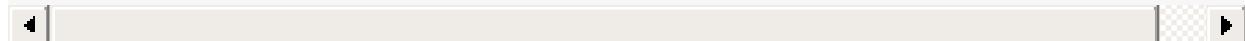
```

class Solution:
 def search(self, nums: List[int], target: int) -> int:
 # first target: find the searching region
 l, r = 0, len(nums)-1

 while l <= r:
 mid = l + (r-l)//2
 if nums[mid] == target:
 return mid

 # to certain it is ascending order
 elif nums[mid] < nums[r]:
 # TODO: Why we need the right bound?
 # Because we have to be certain the target do exist in this range!
 if nums[mid] < target <= nums[r]: # 5 6 7 8 9 1 3, 1
 l = mid+1
 else:
 r = mid-1 # the case that target is not in ascending order side
 else:
 if nums[mid] > target and target >= nums[l]:
 r = mid-1
 else:
 l = mid+1
 return -1

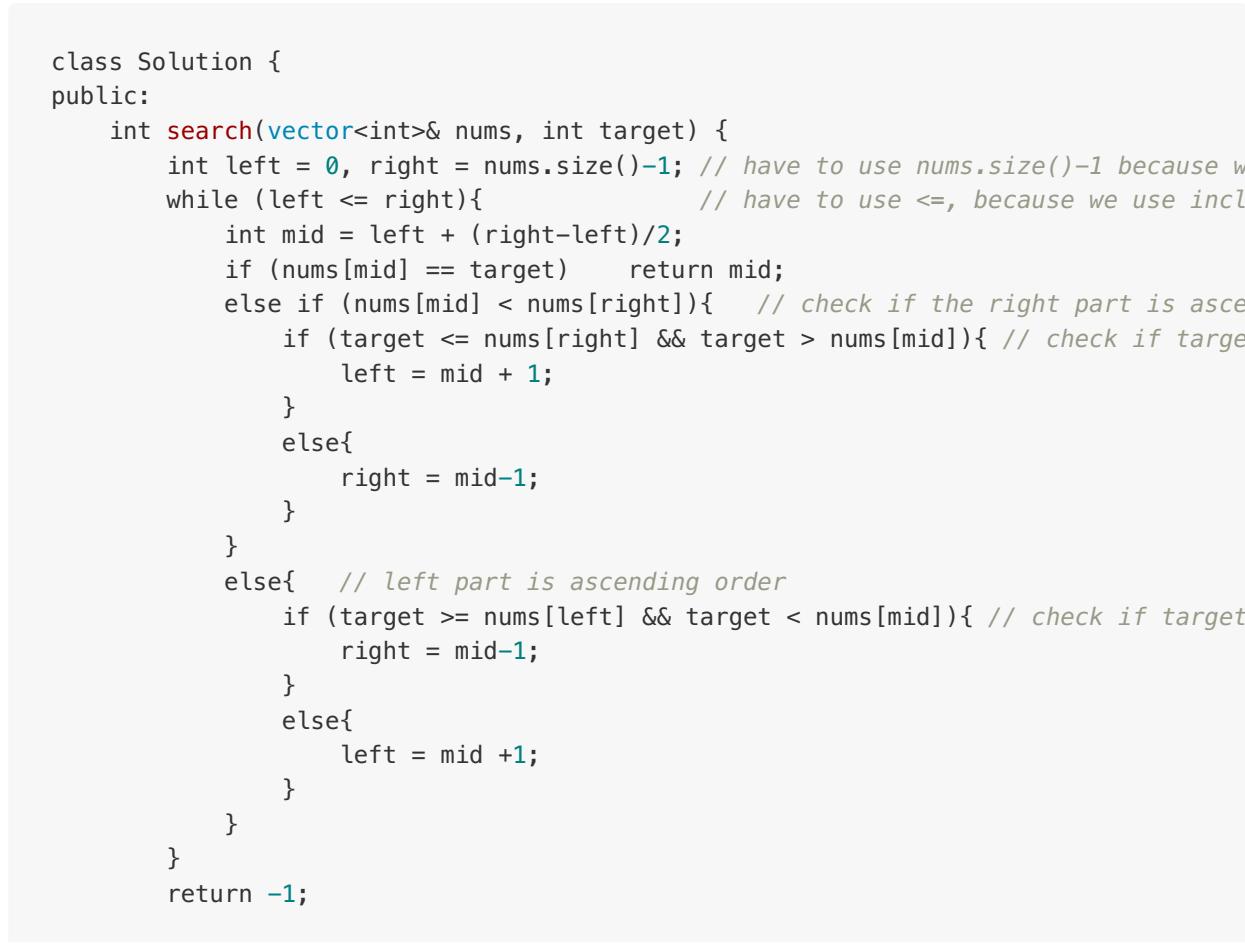
```



```

class Solution {
public:
 int search(vector<int>& nums, int target) {
 int left = 0, right = nums.size()-1; // have to use nums.size()-1 because w
 while (left <= right){ // have to use <=, because we use incl
 int mid = left + (right-left)/2;
 if (nums[mid] == target) return mid;
 else if (nums[mid] < nums[right]){ // check if the right part is asce
 if (target <= nums[right] && target > nums[mid]){ // check if targe
 left = mid + 1;
 }
 else{
 right = mid-1;
 }
 }
 else{ // left part is ascending order
 if (target >= nums[left] && target < nums[mid]){ // check if target
 right = mid-1;
 }
 else{
 left = mid +1;
 }
 }
 }
 return -1;
}

```



```
 }
};
```

## 81. Search in Rotated Sorted Array II | 6/17

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., [0,0,1,2,2,5,6] might become [2,5,6,0,0,1,2]).

You are given a target value to search. If found in the array return true, otherwise return false.

### Example 1:

```
Input: nums = [2,5,6,0,0,1,2], target = 0
Output: true
```

### Example 2:

```
Input: nums = [2,5,6,0,0,1,2], target = 3
Output: false
```

### Follow up:

- This is a follow up problem to [Search in Rotated Sorted Array](#), where `nums` may contain duplicates.
- Would this affect the run-time complexity? How and why?

## 思路

the approach we use here is to find the ascending part  
but the duplicate elem will affect this search approach  
like the case [3,1,1], target=3 will return false,  
because it will assume the left part is ascending, but can't find elem, so it stops and return  
false

Therefore, the direct way here is to remove the duplicate elem which show at leftest and the rightest,  
then our approach work same well here.  
Specifically, we eliminate all of the rightest duplicate item.

## Code

```
class Solution:
 def search(self, nums: List[int], target: int) -> bool:
 l, r = 0, len(nums)-1

 while l <= r:
 mid = l + (r-l)//2
 if nums[mid] == target:
 return True

 elif nums[mid] < nums[r]:
 # right is ascending
 if nums[mid] < target <= nums[r]:
 l = mid + 1
 else:
 r = mid - 1
 elif nums[mid] > nums[r]:
 # left is ascending
 if nums[l] <= target < nums[mid]:
 r = mid - 1
 else:
 l = mid + 1
 else: # nums[mid] == nums[r]
 r -= 1
 return False
```

```
class Solution {
public:
 bool search(vector<int>& nums, int target) {
 int left = 0, right = nums.size()-1; // have to use nums.size()-1 because w
 while (left <= right){ // have to use <=, because we use incl
 int mid = left + (right-left)/2;
 if (nums[mid] == target) return true;
 else if (nums[mid] < nums[right]){ // check if the right part is asce
 if (target <= nums[right] && target > nums[mid]){ // check if targe
 left = mid + 1;
 }
 else{
 right = mid-1;
 }
 }
 else if (nums[mid] > nums[right]){ // left part is ascending order
 if (target >= nums[left] && target < nums[mid]){ // check if target

```

```
 right = mid-1;
 }
 else{
 left = mid +1;
 }
}
else{
 --right; // elemenate the rightest duplicate item until all gone
}
}
return false;
}
};
```

## 153. Find Minimum in Rotated Sorted Array | 6/17

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2]).

Find the minimum element.

You may assume no duplicate exists in the array.

### Example 1:

**Input:** [3,4,5,1,2]  
**Output:** 1

### Example 2:

**Input:** [4,5,6,7,0,1,2]  
**Output:** 0

## 思路

We can find out the smallest elem at the reverting point or the decending point

like [2,3,4,5,1], from 2~5 is ascending, but 5,1 is the steep, the cliff

Then that is where the local max, min happened.

This rule is acceptable except [1,2,3,4,5], so return nums[0] if we can't find cliff.

Code:

right will be accessed, so the initialize should be n-1

mid should be included in next iteration, so right = mid

## Code

```
class Solution {
public:
 int findMin(vector<int>& nums) {
 int left = 0, right = nums.size()-1, n = nums.size();
 while (left < right){
 int mid = left + (right-left)/2;
 if (mid+1 < n && nums[mid] > nums[mid+1]){ // nums[mid+1] is always larger than mid
 return nums[mid+1];
 }
 else if (nums[mid] > nums[right]) { // the min falls on right
 left = mid + 1;
 }
 else{
 right = mid; // should include the mid point; otherwise will fail a
 }
 }
 return nums[0];
 };
};
```

## 154. Find Minimum in Rotated Sorted Array II | 6/17

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2]).

Find the minimum element.

The array may contain duplicates.

## Example 1:

**Input:** [1,3,5]

**Output:** 1

## Example 2:

**Input:** [2,2,2,0,1]

**Output:** 0

## Note:

- This is a follow up problem to [Find Minimum in Rotated Sorted Array](#).
- Would allow duplicates affect the run-time complexity?  
How and why?

## 思路

The follow-up of 153, the solution is same as 81(follow-up of 33)

Duplicate item will destroy the judgement of left or right,  
therefore we eliminate the duplicate item

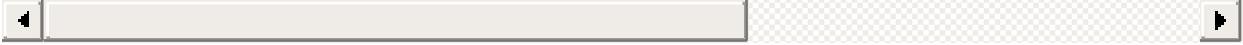
## Code

```
class Solution {
public:
 int findMin(vector<int>& nums) {
 int left = 0, right = nums.size()-1, n = nums.size();
 while (left < right){
 int mid = left + (right-left)/2;
 if (mid+1 < n && nums[mid] > nums[mid+1]){ // nums[mid+1] is always larger than nums[mid]
 return nums[mid+1];
 }
 else if (nums[mid] > nums[right]) { // the min falls on right
 left = mid + 1;
 }
 else if (nums[mid] < nums[left]){

```

```
 right = mid; // should include the mid point; otherwise will fail a
 }
 else right--;
}
return nums[0];
};


```



---

## 162. Find Peak Element | 6/18

---

A peak element is an element that is greater than its neighbors.

Given an input array `nums`, where `nums[i] ≠ nums[i+1]`, find a peak element and return its index.

The array may contain multiple peaks, in that case return the index to any one of the peaks is fine.

You may imagine that `nums[-1] = nums[n] = -∞`.

### Example 1:

**Input:** nums = [1,2,3,1]

**Output:** 2

**Explanation:** 3 is a peak element and your function should return the index number 2.

### Example 2:

**Input:** nums = [1,2,1,3,5,6,4]

**Output:** 1 or 5

**Explanation:** Your function can return either index number 1 where the peak element is 2, or index number 5 where the peak element is 6.

### Note:

Your solution should be in logarithmic complexity.

## 思路

The peak might happen at the location where the number "start" to descend in an ascending order sorted array

thus, we should find the n where is  $\text{nums}[n] > \text{nums}[n+1]$

However, we should care about the border of the array

## Code

```
class Solution {
public:
 int findPeakElement(vector<int>& nums) {
 if (nums.size() == 1) return 0;
 else{
 for (int i=0; i<nums.size()-1; i++){ // shouldn't let let nums[i] = last
 if (nums[i] > nums[i+1]) return i;
```

```
 }
 return nums.size() - 1; // assumed nums[n] = INT_MIN by question
 }
};
```

## 852. Peak Index in a Mountain Array | 6/18

Let's call an array A a mountain if the following properties hold:

A.length >= 3

There exists some  $0 < i < A.length - 1$  such that  $A[0] < A[1] < \dots A[i-1] < A[i] > A[i+1] > \dots > A[A.length - 1]$

Given an array that is definitely a mountain, return any  $i$  such that  $A[0] < A[1] < \dots A[i-1] < A[i] > A[i+1] > \dots > A[A.length - 1]$ .

### Example 1:

**Input:** [0, 1, 0]

**Output:** 1

### Example 2:

**Input:** [0, 2, 1, 0]

**Output:** 1

### Note:

1.  $3 \leq A.length \leq 10000$
2.  $0 \leq A[i] \leq 10^6$
3. A is a mountain, as defined above.

思路

just another way to describe the sorted array peak question like 162

## Code

```
class Solution {
public:
 int peakIndexInMountainArray(vector<int>& A) {
 for (int i=0; i < A.size(); i++){
 if (A[i] > A[i+1]) return i;
 }
 return -1;
 }
};
```

## 69. Sqrt(x) | 6/18

Implement int sqrt(int x).

Compute and return the square root of x, where x is guaranteed to be a non-negative integer.

Since the return type is an integer, the decimal digits are truncated and only the integer part of the result is returned.

### Example 1:

**Input:** 4

**Output:** 2

### Example 2:

**Input:** 8

**Output:** 2

**Explanation:** The square root of 8 is 2.82842...,  
and since

the decimal part is truncated, 2 is  
returned.

## 思路

Back to the native method of finding  $\sqrt{x}$  is  $n^2 = x$ ,  
and keep trying the nearest n until  $n^2$  is next to it.

Btw. the question asks us to find nearest number which no larger than x

Therefore, the question can become find the largest n satisfied  $n^2 \leq x$

We can use Binary search to approach the nearest n.

## Code

```
class Solution {
public:
 int mySqrt(int x) {
 if (x<=1) return x;
 int left = 0, right = x;
 while (left < right){
 int mid = left + (right-left)/2;
 if (x/mid == mid) return mid;
 else if (x/mid > mid){
 left = mid+1;
 }
 else {
 right = mid;
 }
 }
 return right-1; // testing with simple testcase to decide whether minus one
 }
};
```

## 74. Search a 2D Matrix | 7/2

Write an efficient algorithm that searches for a value in an  $m \times n$  matrix. This matrix has the following properties:

Integers in each row are sorted from left to right.

The first integer of each row is greater than the last integer of the previous row.

### Example 1:

**Input:**

```
matrix = [
 [1, 3, 5, 7],
 [10, 11, 16, 20],
 [23, 30, 34, 50]
]
```

target = 3

**Output:** true

### Example 2:

**Input:**

```
matrix = [
 [1, 3, 5, 7],
 [10, 11, 16, 20],
 [23, 30, 34, 50]
]
```

target = 13

**Output:** false

## 思路

It is S-shape sequence.

We can turn 2-D Matrix Search into 1-D by adopt this rule:

$\text{arr}[i/n][i\%n] \Rightarrow$  number i element in 1D

1. left = 0, right = n
2. left < right
3. left = mid + 1, right = mid

## Code

```
class Solution {
public:
 bool searchMatrix(vector<vector<int>>& matrix, int target) {
 if (matrix.empty() || matrix[0].empty()) return false;
 if (target < matrix[0][0] || target > matrix.back().back()) return false;
 int m = matrix.size(), n = matrix[0].size();

 int left = 0, right = m*n;
 while (left < right) {
 int mid = left + (right - left)/2;
 if (matrix[mid / n][mid % n] == target) return true;
 else{
 if (matrix[mid / n][mid % n] < target) {
 left = mid+1;
 }
 else {
 right = mid;
 }
 }
 }
 return false;
 }
};
```

---

## 378. Kth Smallest Element in a Sorted Matrix | 7/2

Given a  $n \times n$  matrix where each of the rows and columns are sorted in ascending order, find the  $k$ th smallest element in the matrix.

Note that it is the  $k$ th smallest element in the sorted order, not the  $k$ th distinct element.

## Example:

```
matrix = [
 [1, 5, 9],
 [10, 11, 13],
 [12, 13, 15]
],
k = 8,

return 13.
```

## Note:

You may assume k is always valid,  $1 \leq k \leq n^2$ .

## 思路

This question is different from above, it is not S-shape Sequence.

Therefore, we cannot use 2D to 1D array transformation.

Solution A uses priority\_queue to deal with this problem,  
because it is definitely a priority\_queue properties question.  
priority\_queue in c++ can automatically sort the queue whenever you push into it.  
This properties can make sure the very top of the queue is the largest element in every loop,  
then we use size limitation to find the Top-k elements in the array.

## Code

Sol A, Priority Queue:

```
class Solution {
public:
 int kthSmallest(vector<vector<int>>& matrix, int k) {
 priority_queue<int> pq;
 for (int i=0; i<matrix.size(); i++){
 for (int j=0; j<matrix[0].size(); j++){
 pq.push(matrix[i][j]);
 if (pq.size() > k) pq.pop();
 }
 }
 }
}
```

```
 }
 return pq.top();
}
};
```

Sol B, Binary Search:

---

## *[Start of the Linked List]*

---

### Dummy Node

Dummy node 是鏈表問題中一個重要的技巧，中文翻譯叫「啞節點」或者「假人頭結點」。

Dummy node 是一個虛擬節點，也可以認為是標竿節點。

Dummy node 就是在鏈表表頭 head 前加一個節點指向 head，即 dummy  $\rightarrow$  head。

Dummy node 的使用多針對單向鏈表沒有前向指標的問題，保證鏈表的 head 不會在刪除操作中遺失。

除此之外，還有一種用法比較少見，就是使用 dummy node 來進行head的刪除操作，

比如 Remove Duplicates From Sorted List II，

一般的方法 current = current.next 是無法刪除 head 元素的，所以這個時候如果有一個dummy node在head的前面。

所以，當鏈表的 head 有可能變化（被修改或者被刪除）時，使用 dummy node 可以簡化程式碼及很多邊界情況的處理，最終返回 dummy.next 即新的鏈表。

---

## 2. Add Two Numbers | 7/3

---

You are given two non-empty linked lists representing two non-negative integers. 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.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

## Example:

**Input:** (2 → 4 → 3) + (5 → 6 → 4)

**Output:** 7 → 0 → 8

**Explanation:** 342 + 465 = 807.

## 思路

We consider to use dummy head if the head of the list may modify or delete.

We don't do any action on dummy, instead we do that on the pointer cur.

We need to consider the imbalance digits situation between two list,

if the digit not exist we assign it with zero.

We count the carry with /10, and count the result with %10.

Finally, move forward the cur, and the two digits.

Note that we should check the digit exist before moving two lists.

Think about the edge case:

99 + 01 = 901 // Carry

99 + 1 = 99 + 10 = 001 // Unequal count between two list

[] + 99 = 00 + 99 = 99 // Empty list

## Code

```
/*
 * Definition for singly-linked list.
 * struct ListNode {
 * int val;
 * ListNode *next;
 * ListNode(int x) : val(x), next(NULL) {}
 * };
 */
/*
edge case:
99 + 01 = 901 // Carry
99 + 1 = 99 + 10 = 001 // Unequal count between two list
[] + 99 = 00 + 99 = 99 // Empty list
*/
class Solution {
public:
 ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
 ListNode* dummy = new ListNode(-1); // A ListNode type pointer to a dummy n
```

```

ListNode* cur = dummy; // use a pointer to move tho the list
int carry = 0;
while (l1 || l2){
 int val1 = l1 ? l1->val : 0;
 int val2 = l2 ? l2->val : 0;
 int sum = val1 + val2 + carry;
 carry = sum/10;
 cur -> next = new ListNode(sum % 10);
 cur = cur->next; // cur move to next
 list move next
 if (l1) l1 = l1->next;
 if (l2) l2 = l2->next;
}
// deal with edge case1
if (carry == 1) cur->next = new ListNode(1);
return dummy->next;
}
};

```

## 445. Add Two Numbers II | 7/3

You are given two non-empty linked lists representing two non-negative integers.

The most significant digit comes first and each of their nodes contain a single digit.

Add the two numbers and return it as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Follow up:

What if you cannot modify the input lists? In other words, reversing the lists is not allowed.

### Example:

**Input:** (7 -> 2 -> 4 -> 3) + (5 -> 6 -> 4)

**Output:** 7 -> 8 -> 0 -> 7

## 思路

If we reverse the linked list, this question is as same as the #2.

However, we use stack to solve this question.

We first push all the node into each stack, cuz stack operation is like the plus op.  
Different here is we have to insert from the front.

To implement insert from the front, we use the concept of swap.

res is the list we keep growing from the front.

We let head->next = res, then res = head.

By doing this, res will add the head->val at the front of it every loop.

We now can let head->val be the carry, if it is existed, to deal with  $99 + 1 = 100$

Note that, we don't want 098 as the answer

, we need to let output ignore the head in this case by return res->next.

Edge case:

$[1] + [2,3,4] = [1,0,0] + [2,3,4] = [3,3,4]$

$[1] + [9,9] = [1,0] + [9,9] = [1,0,9]$

## Code

```
class Solution {
public:
 ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
 stack<int> s1, s2;
 while (l1){
 s1.push(l1->val);
 l1 = l1->next;
 }
 while (l2){
 s2.push(l2->val);
 l2 = l2->next;
 }
 int sum = 0;
 ListNode* res = new ListNode(0);
 while (!s1.empty() || !s2.empty()){
 if (!s1.empty()) { sum += s1.top(); s1.pop(); }
 if (!s2.empty()) { sum += s2.top(); s2.pop(); }
 res->val = sum%10;
 ListNode* head = new ListNode(sum/10); // node to insert from front (培
 head->next = res; // put the whole body of answer after the head node (放
 res = head; // swap it (佔為己有)
 sum /= 10; // keep the carry to next round
 }
 return res->val == 0 ? res->next : res;
 }
};
```

## 24. Swap Nodes in Pairs | 7/4

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

You may not modify the values in the list's nodes, only nodes itself may be changed.

Example:

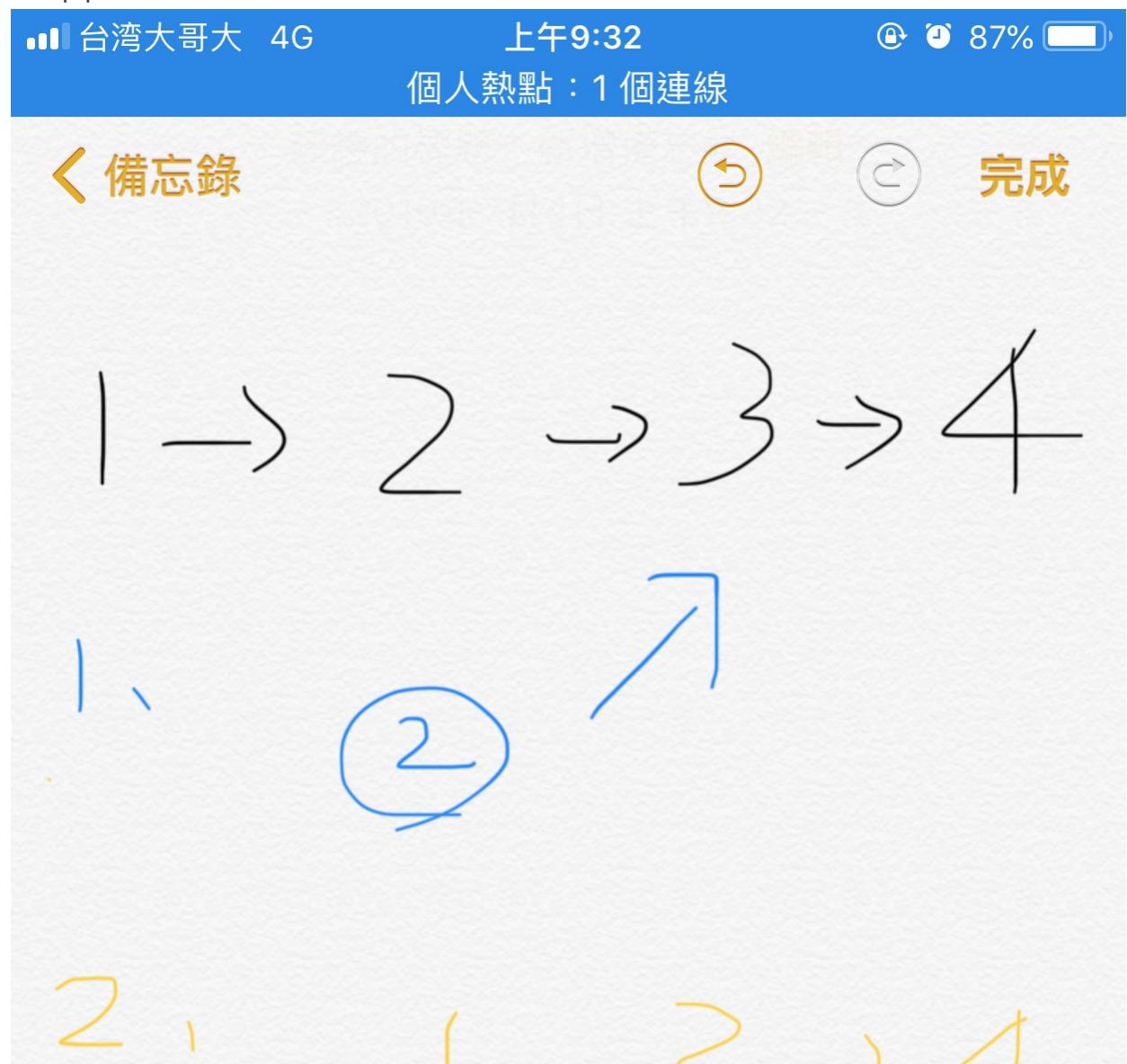
Given  $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ , you should return the list as  $2 \rightarrow 1 \rightarrow 4 \rightarrow 3$ .

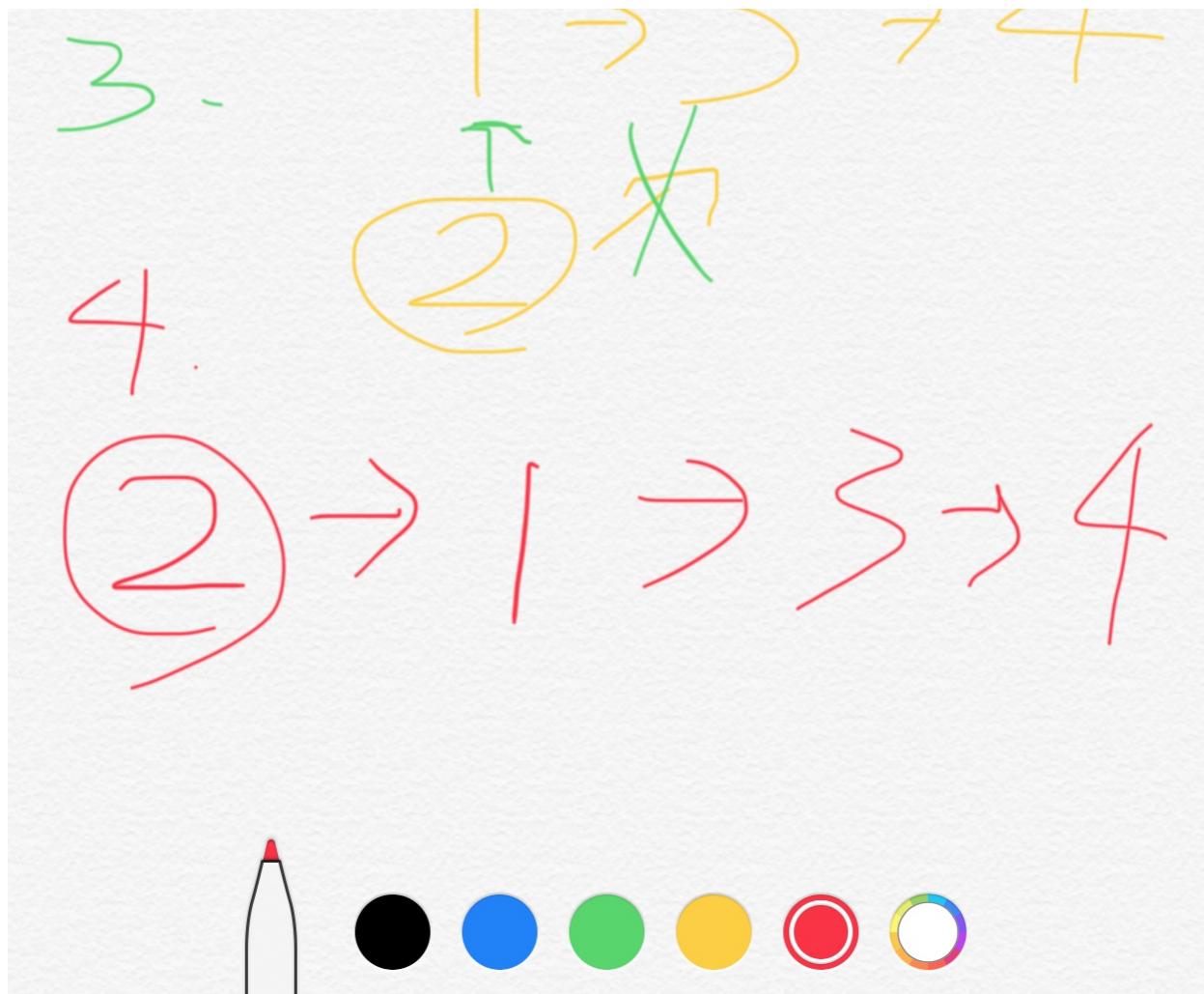
### 思路

We need to modify the head, therefore use a dummy head.

We used pre as the cursor, pre->next and pre->next->next is the target to swap.

Swap process is as below:





Plus step 5, move the cursor to next swap target.

## Code

```

/*
 * Definition for singly-linked list.
 * struct ListNode {
 * int val;
 * ListNode *next;
 * ListNode(int x) : val(x), next(NULL) {}
 * };
 */
/*
node temp
temp = head -> next
head -> next = temp -> next
temp -> next = head
head = temp
... /move the cursor
*/
class Solution {
public:

```

```

ListNode* swapPairs(ListNode* head) {
 ListNode* dummy = new ListNode(-1);
 ListNode* pre = dummy; // pre->next == head
 dummy->next = head;
 while (pre->next && pre->next->next){
 ListNode* temp = pre->next->next; // temp = 2
 pre->next->next = temp -> next; // 1.next = 3
 temp -> next = pre->next; // 2.next = 1
 pre->next = temp; // head = 2
 pre = temp->next; // move the cursor
 }
 return dummy->next;
}

```

## 206. Reverse Linked List | 8/13

Reverse a singly linked list.

### Example:

**Input:** 1->2->3->4->5->NULL  
**Output:** 5->4->3->2->1->NULL

### Follow up:

A linked list can be reversed either iteratively or recursively. Could you implement both?

## 思路

二刷：

Python's recursive is much easier!

Iterative是必須在這個loop更新此節點跟下個節點

Recursive的話只需要交換位置就可以了！

Recursive is much harder to imagine with.

1. Keep asking the next node to give me the reverse to myself.
2. If bump into the situation which no one answer it, he will start to return himself.
3. If someone receive the return reply, he'll ask that adding the next pointer of his next, to point to him.
4. e.g. We are now at 3, 3 recieve the return from 4, which is itself, 'then' it operates like: 1->2->3->4 --> NULL => 1->2->3 <-> 4 NULL
5. But remember, we still have the link 3 -> 4, we need to release it
6. Return the reverse list results to the upper caller.

## Code

Iterative:

```
class Solution {
public:
 ListNode* reverseList(ListNode* head) {
 ListNode* newHead = NULL;
 while (head){
 // Record the node that will be replaced
 // the head will be replaced, and we'll lose the link of the next node
 ListNode* temp = head->next;
 head->next = newHead;
 newHead = head;
 head = temp; // the next iteration
 }
 return newHead;
 }
};
```



Recursively:

```
class Solution {
public:
 ListNode* reverseList(ListNode* head) {
 if (!head || !head->next) return head; // if no one is your next, then release me
 ListNode* newHead = reverseList(head->next);
 head->next->next = head; // POINT your NEXT, to me
 head->next = NULL; // I'm now releasing you
 return newHead;
 }
};
```



....  
Solved: CAN YOU DO IT WITHOUT RETURN A NEWHEAD, RETURN THE ORIGINAL HEAD?  
....

```
def reverseList(self, head: ListNode) -> ListNode:
 # 1->2->3->4->5
 newHead = None
 while head:
 temp = head.next # record 2
 head.next = newHead # 1->NULL
 newHead = head # newHead = 1 (the next's parent)
 head = temp # head = 2
 return newHead
```

```
def recur_reverseList(self, head, prev):
 if not head:
 return prev

 nex = head.next
 head.next = prev
 return recur_reverseList(nex, head) # Solved: CAN I WRITE IT DIFFERENTLY? Only

OR Write it as:
head = self.recur_reverseList(nex, head)
print("started to return:", head.val)
return head
....
 started to return: 5
 started to return: 5
 started to return: 5
 started to return: 5
 started to return: 5
....
if __name__ == '__main__':
 recur_reverseList(head, None)
```

## 141. Linked List Cycle | 7/4

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

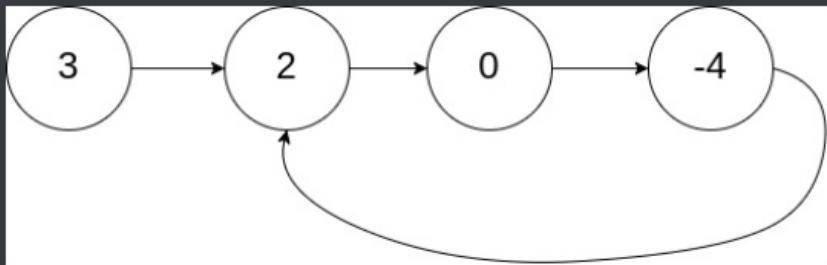
To represent a cycle in the given linked list, we use an integer pos which represents the position (0-indexed) in the linked list where tail connects to. If pos is -1, then there is no cycle in the linked list.

### Example 1:

**Input:** head = [3,2,0,-4], pos = 1

**Output:** true

**Explanation:** There is a cycle in the linked list, where tail connects to the second node.

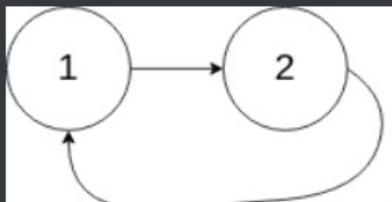


### Example 2:

**Input:** head = [1,2], pos = 0

**Output:** true

**Explanation:** There is a cycle in the linked list, where tail connects to the first node.



### Example 3:

**Input:** head = [1], pos = -1

**Output:** false

**Explanation:** There is no cycle in the linked list.



## 思路

利用快慢指針，如果有環，快指針一定會追上慢指針

while( fast && fast->next ) , why use fast and fast->next?

if we want to write something, we need to make sure it exist.

we write fast at " fast = "

and write fast->next at "fast->next->next"

## Code

```
class Solution {
public:
 bool hasCycle(ListNode* head) {
 ListNode* slow = head;
 ListNode* fast = head;
 while (fast && fast->next) {
 slow = slow->next;
 fast = fast->next->next;
 if (slow == fast) return true;
 }
 return false;
 }
};
```

## 142. Linked List Cycle II | 7/5

Given a linked list, return the node where the cycle begins. If there is no cycle, return null.

To represent a cycle in the given linked list, we use an integer pos which represents the position (0-indexed) in the linked list where tail connects to. If pos is -1, then there is no cycle in the linked list.

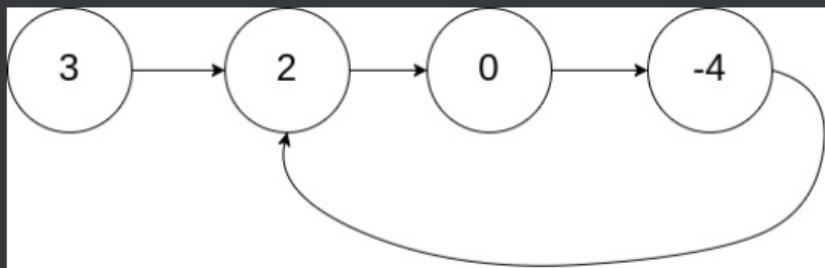
Note: Do not modify the linked list.

### Example 1:

**Input:** head = [3,2,0,-4], pos = 1

**Output:** tail connects to node index 1

**Explanation:** There is a cycle in the linked list,  
where tail connects to the second node.

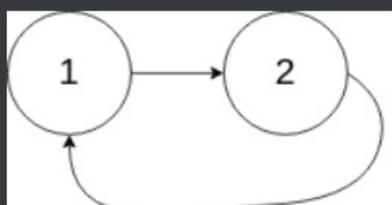


### Example 2:

**Input:** head = [1,2], pos = 0

**Output:** tail connects to node index 0

**Explanation:** There is a cycle in the linked list,  
where tail connects to the first node.



### Example 3:

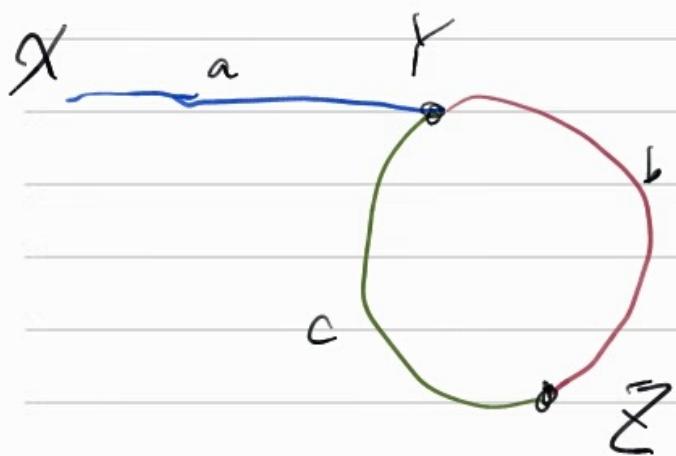
**Input:** head = [1], pos = -1

**Output:** no cycle

**Explanation:** There is no cycle in the linked  
list.



## 思路



$$\begin{aligned} \text{S: } & a + b \\ \text{F: } & a + b + c + b \rightarrow 2(a+b) = a+b+c+b \\ & \boxed{\Rightarrow a=c} \end{aligned}$$

長度： $b+c=a+b$  到第一次相遇 slow 走的距離

假設快慢指針第一次相遇在Z點，  
根據推導就可以打出  $a = c$ ，

利用這個性質我們將slow拉回原點出發，fast則停留在Z點  
兩者一次走一步，第一次相遇的點便是環開始的位置。

遇到Y前的一個節點的next將他等於NULL，就解除環了。

透過  $a = c$  這個性質我們還能求出環的長度。

[詳細](#)

## Code

```
class Solution {
public:
 ListNode* detectCycle(ListNode* head) {
 ListNode* slow = head;
 ListNode* fast = head;
 while (true){
 if (!fast || !fast->next) return NULL;
 fast = fast->next->next;
 slow = slow->next;
 if (fast == slow) break; // Point Z
 }
 // fast and slow both stop at point Z
 // bcuz a = c, we let fast start from point Z
 // and slow start from X
 // the point they met is the point Y, where the answer is
 slow = head;
 while (fast != slow){
 slow = slow->next;
 fast = fast->next;
 }
 return fast; // or slow
 }
};
```

## 21. Merge Two Sorted Lists | 7/5

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.

Example:

Input: 1->2->4, 1->3->4

Output: 1->1->2->3->4->4

# 思路

Understand with Solution1 (Iterative Way). And learn to write Solution2 (Recursive Way).

## Code

Solution1:

```
class Solution {
public:
 ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
 ListNode* dummy = new ListNode(-1);
 ListNode* cur = dummy;
 while (l1 && l2) {
 if (l1->val < l2->val) {
 cur -> next = l1;
 l1 = l1->next;
 } else{
 cur -> next = l2;
 l2 = l2->next;
 }
 cur = cur->next;
 }
 // Remain List
 if (l1) cur -> next = l1;
 if (l2) cur -> next = l2;
 return dummy->next;
 }
};
```

Solution2:

```
class Solution {
public:
 ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
 if (!l1) return l2;
 if (!l2) return l1;
 if (l1->val < l2->val) {
 l1->next = mergeTwoLists(l1->next, l2);
 return l1;
 } else {
 l2->next = mergeTwoLists(l1, l2->next);
 return l2;
 }
 }
};
```

## 23. Merge k Sorted Lists | 7/5

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

### Example:

#### Input:

```
[
 1->4->5,
 1->3->4,
 2->6
]
```

**Output:** 1->1->2->3->4->4->5->6

## 思路

- Sol 1: Merge two list is easier to solve, so here we use divide and conquer to do.

HOW TO DIVIDE?

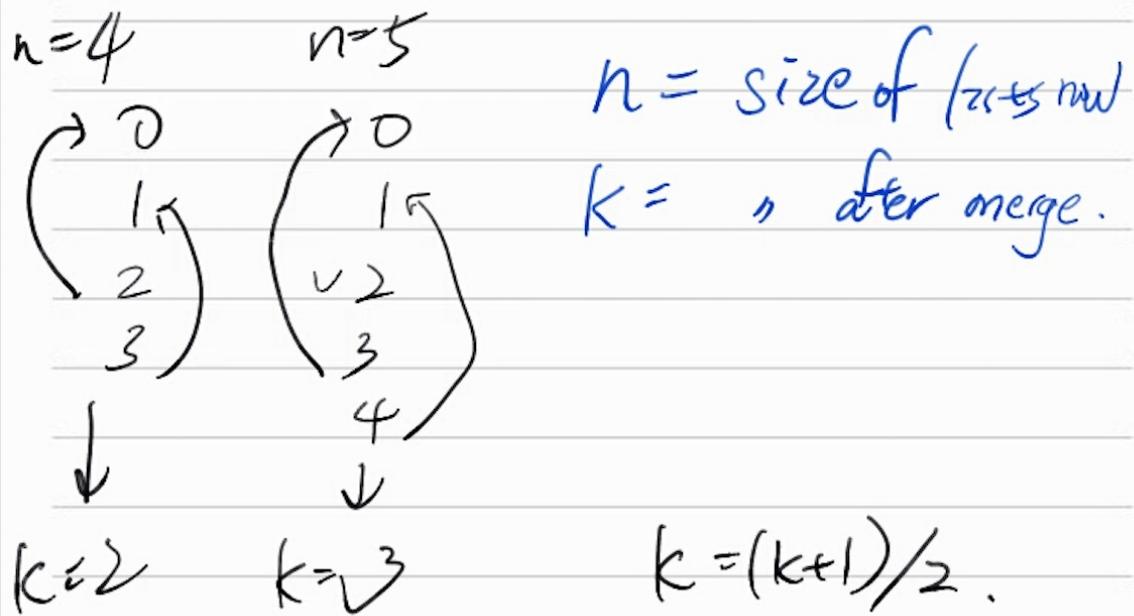
We did DIVIDE and MERGE together here.

We don't want to use any other space for storing merged data.

So, who split it, is the whom will store it.

We calculate the size of the lists after doing this process. (int k)

And use this size as the iterating numbers for split and merge.



- Sol 2: Priority Queue

We don't push all the nodes into the queue one times, we push the head of the list instead.

See this discussion for why.

reddyinand604 ★ 4 April 10, 2019 2:07 AM  
 if(tail->next) q.push(tail->next);  
 I didn't understand these line why we are checking if tail->next is not null and pushing it to queue again  
 ▲ 1 ▾ Hide 3 replies Reply

Azkaban153 ★ 0 Last Edit: April 11, 2019 11:17 PM  
 Of course you can. But then the time complexity would be  $O(n^2)$  for pushing all the nodes first and then popping from queue. By implementing it this way, it's  $O(n)$ .  
 ▲ 1 ▾ Reply

reddyinand604 ★ 4 Last Edit: April 11, 2019 11:05 PM  
 Thank you I have a doubt can we push all the nodes in the k lists at once to queue and start popping elements  
 ▲ 0 ▾ Reply

Azkaban153 ★ 0 Last Edit: April 11, 2019 10:52 PM  
 tail is pointing at the next plausible node in some linked list. If we've reached the end of that list (tail == NULL), we won't be pushing it in the priority queue. Otherwise, there are nodes left in that list that need to be merged. Hence, we push them to queue.  
 ▲ 0 ▾ Reply

## Code

Solution1 (Divide and Conquer) beat 99%:

```

class Solution {
public:
 ListNode* mergeKLists(vector<ListNode*>& lists) {
 if (lists.empty()) return NULL;
 }
}

```

```

 int n = lists.size(), k = n;
 while (n > 1){
 k = (k+1) / 2;
 for (int i=0; i < n/2; i++){
 lists[i] = mergeTwoLists(lists[i], lists[i+k]);
 }
 n = k; // next merge iteration
 }
 return lists[0];
 }

 ListNode* mergeTwoLists(ListNode* a, ListNode* b){
 ListNode* dummy = new ListNode(-1);
 ListNode* cur = dummy;
 while (a && b) {
 if (a->val < b->val){
 cur->next = a;
 a = a->next;
 }
 else {
 cur->next = b;
 b = b->next;
 }
 cur = cur->next;
 } // end if one lists is end
 // add remain nodes
 if (a) cur->next = a;
 if (b) cur->next = b;
 return dummy->next;
 }
};

```

Another Divide and Conquer way

```

class Solution {
public:
 ListNode* mergeKLists(vector<ListNode*>& lists) {
 return helper(lists, 0, (int)lists.size() - 1);
 }
 ListNode* helper(vector<ListNode*>& lists, int start, int end) {
 if (start > end) return NULL;
 if (start == end) return lists[start];
 int mid = start + (end - start) / 2;
 ListNode* left = helper(lists, start, mid);
 ListNode* right = helper(lists, mid + 1, end);
 return mergeTwoLists(left, right);
 }
 ListNode* mergeTwoLists(ListNode* a, ListNode* b){
 ... // same as above
 }
};

```

Solution2 (priority\_queue):

```
class Solution {
public:
 ListNode* mergeKLists(vector<ListNode*>& lists) {
 auto cmp = [] (ListNode* &a, ListNode* &b){
 return a->val > b->val;
 };
 priority_queue<ListNode*, vector<ListNode*>, decltype(cmp) > q(cmp);

 // only push the head of each list, cuz of the efficiency
 for (auto node : lists) {
 if (node) q.push(node);
 }

 ListNode* dummy = new ListNode(-1);
 ListNode* cur = dummy;
 while (!q.empty()){
 auto t = q.top(); q.pop();
 cur->next = t;
 cur = cur->next;
 if (cur->next) q.push(cur->next); // push the next possible smaller number
 }
 return dummy->next;
 }
};
```

## 147. Insertion Sort List | 7/8

Sort a linked list using insertion sort.

6 5 3 1 8 7 2 4

6 5 3 1 8 7 2 4

Algorithm of Insertion Sort:

Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list.

At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there.

It repeats until no input elements remain.

### Example 1:

**Input:** 4->2->1->3

**Output:** 1->2->3->4

### Example 2:

**Input:** -1->5->3->4->0

**Output:** -1->0->3->4->5

## 思路

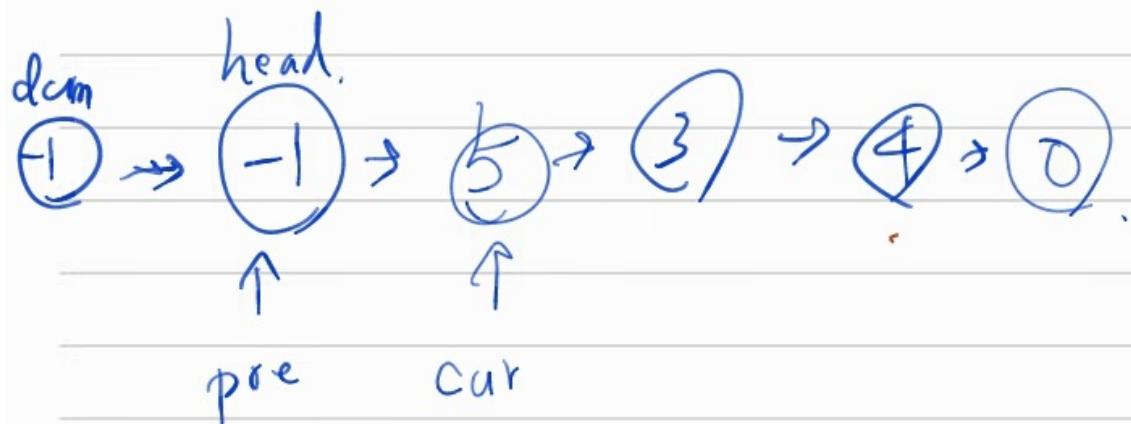
cur is the cursor to move tho the list.

pre is the cursor to find the position to insert.

cur.val should always < cur->next.val, OR we will exchange.

pre will find a pos, where the pre->next.val > inserting value.

-1 → 5 → 3 → 4 → 0



$\text{temp} = \text{pre} \rightarrow \text{next}$

$\text{pre} \rightarrow \text{n} = 3$ .  $\text{---} \rightarrow \text{3} \rightarrow \text{5} \rightarrow \text{4}$

$\text{cur} \rightarrow \text{n} = \text{cur} \rightarrow \text{n} \rightarrow \text{n}$ .

$\text{pre} \rightarrow \text{h} \rightarrow \text{n} = \text{temp}$ .

$\text{pre} = \text{head}$ . (move to begin).

Code

```
class Solution {
public:
```

```

ListNode* insertionSortList(ListNode* head) {
 ListNode* dummy = new ListNode(-1);
 dummy->next = head;
 ListNode* pre = dummy;
 ListNode* cur = dummy->next;

 while (cur) {
 // value exchange
 if (cur->next && cur->next->val < cur->val){ // cur->next should be rel
 // locating the pos to insert in from head
 while (pre->next && pre->next->val < cur->next->val) {
 pre = pre->next;
 } // pre -> next is the pos to insert in

 ListNode* temp = pre->next; // original node
 pre->next = cur->next;
 cur->next = cur->next->next;
 pre->next->next = temp;
 pre = dummy; // return pre to the front
 }
 else { // cur iterating
 cur = cur->next;
 }
 }
 return dummy->next;
}

```



## 148. Sort List | 7/9

Sort a linked list in  $O(n \log n)$  time using constant space complexity.

### Example 1:

**Input:** 4->2->1->3

**Output:** 1->2->3->4

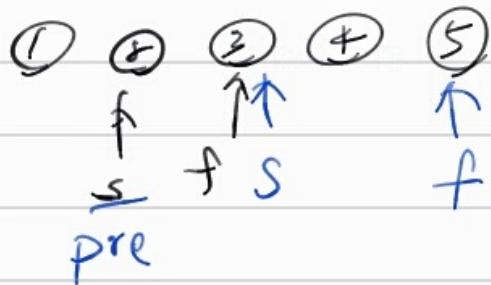
### Example 2:

**Input:** -1->5->3->4->0

**Output:** -1->0->3->4->5

## 思路

奇



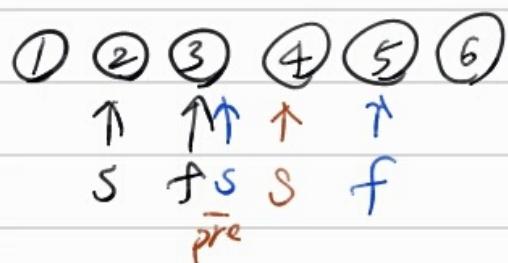
pre → ②

slow → ③

fast → ⑤

快慢指針

偶



pre → ③

slow → ④

fast,

NUL

切中

叉

Next:  $\text{pre} \rightarrow \text{next} = \text{NULL}$

得到 2 个 List: { head: ① → ② → ③ → NULL  
slow: ④ → ⑤ → ⑥ → NULL }

常見排序方法有很多，插入排序，選擇排序，堆排序，快速排序，冒泡排序，歸併排序，桶排序等等。。它們的時間複雜度不盡相同，而這裡題目限定了時間必須為  $O(n \lg n)$ ，符合要求只有快速

排序，歸併排序，堆排序，而根據單鏈表的特點，最適於用歸併排序。為啥呢？這是由於鏈表自身的特點決定的，由於不能通過坐標來直接訪問元素，所以快排什麼的可能不太容易實現（但是被評論區的大神們打臉，還是可以實現的），堆排序的話，如果讓新建結點的話，還是可以考慮的，若只能交換結點，最好還是不要用。而歸併排序（又稱混合排序）因其可以利用遞歸來交換數字，天然適合鏈表這種結構。

併排序的核心是一個 merge() 函數，其主要是合併兩個有序鏈表，這個在 LeetCode 中也有單獨的題目 Merge Two Sorted Lists。由於兩個鏈表是要有序的才能比較容易 merge，那麼對於一個無序的鏈表，如何才能拆分成有序的兩個鏈表呢？我們從簡單來想，什麼時候兩個鏈表一定都是有序的？就是當兩個鏈表各只有一個結點的時候，一定是有序的。而歸併排序的核心其實是分治法 Divide and Conquer，就是將鏈表從中間斷開，分成兩部分，左右兩邊再分別調用排序的遞歸函數 sortList()，得到各自有序的鏈表後，再進行 merge()，這樣整體就是有序的了。因為子鏈表的遞歸函數中還是會再次拆成兩半，當拆到鏈表只有一個結點時，無法繼續拆分了，而這正好滿足了前面所說的“一個結點的時候一定是有序的”，這樣就可以進行 merge 了。然後再回溯回去，每次得到的都是有序的鏈表，然後進行 merge，直到還原整個長度。這裡將鏈表從中間斷開的方法，採用的就是快慢指針，大家可能對快慢指針找鏈表中的環比較熟悉，其實找鏈表中的中點同樣好使，因為快指針每次走兩步，慢指針每次走一步，當快指針到達鏈表末尾時，慢指針正好走到中間位置

## Code

```
class Solution {
public:
 ListNode* sortList(ListNode* head) {
 if (!head || !head->next) return head; // split the list until rest one part
 ListNode* slow = head;
 ListNode* fast = head;
 ListNode* pre = head;

 while (fast && fast->next){
 pre = slow; // record the last state of slow
 slow = slow->next;
 fast = fast->next->next;
 }
 pre->next = NULL; // So, head~pre->next is front part of list, and slow is
 return merge(sortList(head), sortList(slow));
 }

 ListNode* merge(ListNode* l1, ListNode* l2) {
 if (!l1) return l2;
 if (!l2) return l1;

 if (l1->val < l2->val){
 l1->next = merge(l1->next, l2);
 return l1;
 } else{
 l2->next = merge(l1, l2->next);
 return l2;
 }
 }
}
```

```
};
```

## 707. Design Linked List | 7/11

Design your implementation of the linked list. You can choose to use the singly linked list or the doubly linked list. A node in a singly linked list should have two attributes: val and next. val is the value of the current node, and next is a pointer/reference to the next node. If you want to use the doubly linked list, you will need one more attribute prev to indicate the previous node in the linked list. Assume all nodes in the linked list are 0-indexed.

Implement these functions in your linked list class:

get(index) : Get the value of the index-th node in the linked list. If the index is invalid, return -1.  
addAtHead(val) : Add a node of value val before the first element of the linked list. After the insertion, the new node will be the first node of the linked list.  
addAtTail(val) : Append a node of value val to the last element of the linked list.  
addAtIndex(index, val) : Add a node of value val before the index-th node in the linked list. If index equals to the length of linked list, the node will be appended to the end of linked list. If index is greater than the length, the node will not be inserted.  
deleteAtIndex(index) : Delete the index-th node in the linked list, if the index is valid.

## Example:

```
MyLinkedList linkedList = new MyLinkedList();
linkedList.addAtHead(1);
linkedList.addAtTail(3);
linkedList.addAtIndex(1, 2); // linked list
becomes 1->2->3
linkedList.get(1); // returns 2
linkedList.deleteAtIndex(1); // now the linked
list is 1->3
linkedList.get(1); // returns 3
```

## Note:

- All values will be in the range of  $[1, 1000]$ .
- The number of operations will be in the range of  $[1, 1000]$ .
- Please do not use the built-in LinkedList library.

## 思路

Design your Node structure first.

## Code

```
class MyLinkedList {
private:
 struct Node{
 int val;
 Node* next;
 Node(int x, Node* n): val(x), next(n) {}
 };
 Node* head;
 Node* tail;
 int size;
public:
 /** Initialize your data structure here. */
 MyLinkedList() {
```

```

// Used the structure we define above
head = NULL;
size = 0;
}

/** Get the value of the index-th node in the linked list. If the index is invalid
int get(int index) {
 if (index < 0 || index >= size) return -1;
 Node* cur = head;
 for (int i=0; i < index; i++) cur = cur->next;
 return cur->val;
}

/** Add a node of value val before the first element of the linked list. After
void addAtHead(int val) {
 Node* t = new Node(val, head); // t->val = val, t->next = head
 head = t;
 ++size;
}

/** Append a node of value val to the last element of the linked list. */
void addAtTail(int val) {
 Node* cur = head;
 while (cur->next) cur = cur->next;
 cur->next = new Node(val, NULL);
 ++size;
}

/** Add a node of value val before the index-th node in the linked list. If index is invalid
void addAtIndex(int index, int val) {
 if (index < 0 || index >= size) return;
 else if (index == 0) return addAtHead(val);
 else if (index == size-1) return addAtTail(val);
 else{
 Node* cur = head;
 for (int i=0; i < index-1; i++) cur = cur->next;
 Node* temp = new Node(val, cur->next);
 cur->next = temp;
 ++size;
 }
}

/** Delete the index-th node in the linked list, if the index is valid. */
void deleteAtIndex(int index) {
 if (index < 0 || index >= size) return;
 else{
 Node* dummy = new Node(-1, head);
 Node* cur = dummy;
 for (int i=0; i<index; i++) cur = cur->next;
 // after for-loop deleting target is cur->next
 cur->next = cur->next->next;
 --size;
 }
}
};

```

```
/**
 * Your MyLinkedList object will be instantiated and called as such:
 * MyLinkedList* obj = new MyLinkedList();
 * int param_1 = obj->get(index);
 * obj->addAtHead(val);
 * obj->addAtTail(val);
 * obj->addAtIndex(index,val);
 * obj->deleteAtIndex(index);
 */
```



## [Start of the Array]

---

很少考: 118, 119, 169, 229, 219, 220

需訂閱: 244, 245

---

## 27. Remove Element | 7/11

---

Given an array `nums` and a value `val`, 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 by modifying the input array in-place with O(1) extra memory.

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

### Example 1:

Given `nums` = [3,2,2,3], `val` = 3,

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

It doesn't matter what you leave beyond the returned `length`.

### Example 2:

Given `nums` = [0,1,2,2,3,0,4,2], `val` = 2,

Your function should return `length` = 5, with the first five elements of `nums` containing 0, 1, 3, 0, and 4.

Note that the order of those five elements can be arbitrary.

It doesn't matter what values are set beyond the returned `length`.

## Clarification:

Confused why the returned value is an integer but your answer is an array?

Note that the input array is passed in by **reference**, which means modification to the input array will be known to the caller as well.

Internally you can think of this:

```
// nums is passed in by reference. (i.e., without
// making a copy)
int len = removeElement(nums, val);

// any modification to nums in your function
// would be known by the caller.
// using the length returned by your function, it
// prints the first len elements.
for (int i = 0; i < len; i++) {
 print(nums[i]);
}
```

## 思路

回傳的int會拿來當讀取這個陣列的元素數量

like [1,2,2,2,2,2] Output: [2,2,2,2,2]

like [1,2,2,2,2,1] Output: [2,2,2,2]

## Code

```
class Solution {
public:
 int removeElement(vector<int>& nums, int val) {
```

```
int res = 0;
for (int i=0; i< nums.size(); i++){
 if (nums[i] != val) nums[res++] = nums[i];
}
return res;
};
```

---

## 26. Remove Duplicates from Sorted Array | 7/11

---

Given a sorted array `nums`, 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 by modifying the input array in-place with O(1) extra memory.

### Example 1:

Given `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 returned length.

### Example 2:

Given `nums` = [0,0,1,1,1,2,2,3,3,4],

Your function should return length = 5, with the first five elements of `nums` being modified to 0, 1, 2, 3, and 4 respectively.

It doesn't matter what values are set beyond the returned length.

## Clarification:

Confused why the returned value is an integer but your answer is an array?

Note that the input array is passed in by **reference**, which means modification to the input array will be known to the caller as well.

Internally you can think of this:

```
// nums is passed in by reference. (i.e., without
// making a copy)
int len = removeDuplicates(nums);

// any modification to nums in your function
// would be known by the caller.
// using the length returned by your function, it
// prints the first len elements.
for (int i = 0; i < len; i++) {
 print(nums[i]);
}
```

## 思路

remember to put on the last element in the list  
which is definitely will be miss by comparison  
no matter what the last two element is the same or not

## Code

```
class Solution:
 def removeDuplicates(self, nums: List[int]) -> int:
 if (len(nums) < 2):
```

```

 return len(nums)
 cur, nex = 0, 1
 while nex != len(nums):
 if nums[cur] != nums[nex]:
 nums[cur+1] = nums[nex]
 cur += 1
 else:
 nex += 1
 return cur+1

```

```

class Solution {
public:
 int removeDuplicates(vector<int>& nums) {
 if (nums.empty()) return 0;
 int res = 0;
 for (int i=0; i < nums.size() -1; i++){
 if (nums[i] != nums[i+1]) nums[res++] = nums[i];
 }
 // put on the last element in the last
 nums[res++] = nums[nums.size()-1];
 return res;
 }
};

```

## 80. Remove Duplicates from Sorted Array II | 7/12

Given a sorted array `nums`, remove the duplicates in-place such that duplicates appeared at most twice and return the new length.

Do not allocate extra space for another array, you must do this by modifying the input array in-place with O(1) extra memory.

### Example 1:

Given `nums` = [1,1,1,2,2,3],

Your function should return length = 5, with the first five elements of `nums` being 1, 1, 2, 2 and 3 respectively.

It doesn't matter what you leave beyond the returned length.

### Example 2:

Given `nums` = [0,0,1,1,1,1,2,3,3],

Your function should return length = 7, with the first seven elements of `nums` being modified to 0, 0, 1, 1, 1, 2, 3 and 3 respectively.

It doesn't matter what values are set beyond the returned length.

## Clarification:

Confused why the returned value is an integer but your answer is an array?

Note that the input array is passed in by **reference**, which means modification to the input array will be known to the caller as well.

Internally you can think of this:

```
// nums is passed in by reference. (i.e., without
// making a copy)
int len = removeDuplicates(nums);

// any modification to nums in your function
// would be known by the caller.
// using the length returned by your function, it
// prints the first len elements.
for (int i = 0; i < len; i++) {
 print(nums[i]);
}
```

## 思路

Considering add elem into array, except for different element circumstance, we have to find out the other circumstance to make sure we got twice.

The First  $i == i+1$  is the circumstance we look for, so we use a flag to distinguish it.

## Code

```
class Solution {
public:
 int removeDuplicates(vector<int>& nums) {
 if (nums.empty()) return 0;
```

```

int res=0;
bool isExist = false;
for (int i=0; i < nums.size()-1; i++){
 if (nums[i] != nums[i+1]){
 nums[res++] = nums[i];
 isExist = false;
 }
 else{
 if (!isExist){ // if i == i+1 but is the first two elem
 nums[res++] = nums[i];
 isExist = true;
 }
 }
}
nums[res++] = nums[nums.size()-1]; // add the last elem
return res;
};


```

## 277. Find the Celebrity(\$ Lint: 645) | 7/12

---

Suppose you are at a party with  $n$  people (labeled from 0 to  $n - 1$ ) and among them, there may exist one celebrity. The definition of a celebrity is that all the other  $n - 1$  people know him/her but he/she does not know any of them.

Now you want to find out who the celebrity is or verify that there is not one. The only thing you are allowed to do is to ask questions like: "Hi, A. Do you know B?" to get information of whether A knows B. You need to find out the celebrity (or verify there is not one) by asking as few questions as possible (in the asymptotic sense).

You are given a helper function `bool knows(a, b)` which tells you whether A knows B. Implement a function `int findCelebrity(n)`, your function should minimize the number of calls to `knows`.

### **Example1**

**Input:**

2 // next  $n * (n - 1)$  lines

0 knows 1

1 does not know 0

**Output:** 1

**Explanation:**

Everyone knows 1, and 1 knows no one.

### **Example2**

**Input:**

3 // next  $n * (n - 1)$  lines

0 does not know 1

0 does not know 2

1 knows 0

1 does not know 2

2 knows 0

2 knows 1

**Output:** 0

**Explanation:**

Everyone knows 0, and 0 knows no one.

0 does not know 1, and 1 knows 0.

2 knows everyone, but 1 does not know 2.

### Notice

There will be exactly one celebrity if he/she is in the party.

Return the celebrity's label if there is a celebrity in the party. If there is no celebrity, return -1

### 思路

{ celebrity NOT know anyone.  
celebrity known by everyone.

0 0 0 0 0

0

{ A know B  $\rightarrow$  A is not  
A don't know B  $\rightarrow$  B is not.

$n=3$ .

0, 1, 2. guess 0 is. 0 = fixed

$\text{knows}(0, 1) = \text{false}$ . )  $n-1 \not\models \text{false}$

$\text{knows}(0, 2) = \text{false}$ . )  $n-1 \not\models \text{true}$

$\text{knows}(1, 0) = \text{true}$ . )  $n-1 \not\models \text{true}$

$\text{knows}(2, 0) = \text{true}$ . )  $n-1 \not\models \text{true}$

Code

```
// Forward declaration of the knows API.
bool knows(int a, int b);
```

```

class Solution {
public:

 // * @param n a party with n people
 // * @return the celebrity's label or -1

 int findCelebrity(int n) {
 int cnt_false = 0;
 int cnt_true = 0;
 for (int i=0; i<n; i++){ // named i as celebrity
 // check false
 for (int j=0; j<n; j++){
 if (i==j) continue;
 if (knows(i,j)){ // i knows anyone then i is not
 cnt_false = 0;
 break;
 }
 else cnt_false++;
 }
 if (cnt_false != n-1) continue;

 for (int j=0; j<n; j++){
 if (i==j) continue;
 if (!knows(j,i)){
 cnt_true = 0;
 break;
 }
 else cnt_true++;
 }
 if (cnt_true == n-1) return i;
 }
 return -1;
 }
};

```

---

## 189. Rotate Array | 7/12

---

Given an array, rotate the array to the right by k steps, where k is non-negative.

### Example 1:

**Input:** [1,2,3,4,5,6,7] and k = 3

**Output:** [5,6,7,1,2,3,4]

**Explanation:**

rotate 1 steps to the right: [7,1,2,3,4,5,6]

rotate 2 steps to the right: [6,7,1,2,3,4,5]

rotate 3 steps to the right: [5,6,7,1,2,3,4]

### Example 2:

**Input:** [-1,-100,3,99] and k = 2

**Output:** [3,99,-1,-100]

**Explanation:**

rotate 1 steps to the right: [99,-1,-100,3]

rotate 2 steps to the right: [3,99,-1,-100]

### Note:

- Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem.
- Could you do it in-place with O(1) extra space?

## 思路

為了讓取代的動作能夠持續地進行，  
當我們這個state取代了某個數，  
某個數就是下一輪即將取代別人的數。

1 2 3 4 5 6 7

1 2 3 **1** 5 6 7

1 2 3 1 5 6 **4**

1 2 **7** 1 5 6 4

1 2 7 1 5 **3** 4

1 **6** 7 1 5 3 4

1 6 7 1 **2** 3 4

**5** 6 7 1 2 3 4

How to change the elem?

```
idx -> (idx + k) % n
```

Note: be careful for deadlock like

e.g. [1,2,3,4] k=2

start at idx = 0

i=0: idx = 2

i=1: idx = 0 // we can increase from 0 to 1 to avoid deadlock

i=2: idx = 2

.....

## Code

空間複雜度(1)的方法：

```
class Solution {
public:
 void rotate(vector<int>& nums, int k) {
 if (nums.empty() || k % nums.size() == 0) return;
 int now = 0, next = nums[0], start = 0, idx = 0, n = nums.size();
 for (int i=0; i<n; i++){
 now = next; // assign the replacing value for this round
 idx = (idx+k) % n; // calc the position to be replaced
 next = nums[idx]; // stored the replaced value, which will be used for
 nums[idx] = now; // replacing the value

 // deadlock avoidance
 if (idx == start){
 start++;
 idx = start;
 next = nums[idx];
 }
 }
 }
};
```

時間複雜度低的方法：

```
class Solution {
public:
 void rotate(vector<int>& nums, int k) {
 vector<int> t = nums;
 for (int i = 0; i < nums.size(); ++i)
 nums[(i + k) % nums.size()] = t[i];
 }
};
```

Given an unsorted integer array, find the smallest missing positive integer.

**Example 1:**

Input: [1, 2, 0]

Output: 3

**Example 2:**

Input: [3, 4, -1, 1]

Output: 2

**Example 3:**

Input: [7, 8, 9, 11, 12]

Output: 1

**Note:**

Your algorithm should run in  $O(n)$  time and uses constant extra space.

## 思路

We can't sort or find because of  $O(n)$  time limit.

So we use swap and array's index to solve.

We can know which is the least elem by referring to the index.

value 1 should be put at index = 0, value 5 at idx=4

value k at idx = nums[k-1]

after the positioning, we can find out the least positive int by walking through.

Note: might consider the range of the array otherwise overflowing.

$[3, 4, -1, 1]$

$\text{nums}[i], \text{nums}[\text{nums}[i] - 1]$

$i=0$       3.       $\text{nums}[2]$

$[-1, 4, 3, 1]$ .

$i=1$       4.       $\text{nums}[3]$

$[-1, 1, 3, 4]$ .

$[1, -1, 3, 4]$ .



That's why use 'while', not 'if'

## Code

```
class Solution {
public:
```

```

int firstMissingPositive(vector<int>& nums) {
 int n = nums.size();
 for (int i=0; i<n; i++){
 while (nums[i] > 0 && nums[i] <= n && nums[i] != nums[nums[i]-1]){
 swap (nums[i], nums[nums[i]-1]);
 } // positioning elem which >0 and <n at the 'right' position
 }
 for (int i=0; i<n; i++){
 if (nums[i] != i+1) return i+1;
 }
 return n+1;
}

```

## 299. Bulls and Cows | 7/15

---

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.

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.

Please note that both secret number and friend's guess may contain duplicate digits.

### Example 1:

**Input:** secret = "1807", guess = "7810"

**Output:** "1A3B"

**Explanation:** 1 bull and 3 cows. The bull is 8, the cows are 0, 1 and 7.

### Example 2:

**Input:** secret = "1123", guess = "0111"

**Output:** "1A1B"

**Explanation:** The 1st 1 in friend's guess is a bull, the 2nd or 3rd 1 is a cow.

**Note:** You may assume that the secret number and your friend's guess only contain digits, and their lengths are always equal.

## 思路

We firstly defined the conduct of the secret and guess in the map:

- for secret: We Count it, so used the 'add' operation to define.
- for guess : We Use it, so used the 'minus' operation to define.

Therefore, we can use only one for-loop to solve this question.

After the bull determine, we do the cow determine.

In the i round:

If the number of the secret in the map is smaller than 0,  
means it showed in the guess before. We add the cows.

But don't forget to Count or add to map.

if the `number` of the guess in the map is larger than `0`, means it showed in the secret before. We also `add` the cows. But don't forget to Use or minus to map.

Note:

remind, if  $(k++ > 0)$  , equal to determine if k is larger than 0 and no matter what the result is  $k+=1$  after the determination.

remind, if  $(++k > 0)$  , equal to do  $k+=1$  first, and determine if the  $k+=1$  is larger than 0

## Code

Advanced way:

```
class Solution {
public:
 string getHint(string secret, string guess) {
 int bull = 0, cow = 0;
 int m[10] = {0}; // question assumed all the input is digit
 for (int i=0; i < secret.size(); i++){
 if (secret[i] == guess[i]) bull++;
 else{
 if (m[secret[i]-'0']++ < 0) cow++; // if showed in the guess before
 if (m[guess[i]-'0']-- > 0) cow++; // if showed in the secret before
 }
 }
 return to_string(bull) + "A" + to_string(cow) + "B" ;
 }
};
```

Basic way:

```
class Solution {
public:
 string getHint(string secret, string guess) {
 int m[256] = {0}, bulls = 0, cows = 0;
 for (int i = 0; i < secret.size(); ++i) {
 if (secret[i] == guess[i]) ++bulls;
 else ++m[secret[i]];
 }
 for (int i = 0; i < secret.size(); ++i) {
 if (secret[i] != guess[i] && m[guess[i]]) {
 ++cows;
 --m[guess[i]];
 }
 }
 }
};
```

```
 }
 return to_string(bulls) + "A" + to_string(cows) + "B";
 }
};
```

## 134. Gas Station | 7/16

There are N gas stations along a circular route, where the amount of gas at station i is  $\text{gas}[i]$ .

You have a car with an unlimited gas tank and it costs  $\text{cost}[i]$  of gas to travel from station i to its next station ( $i+1$ ). You begin the journey with an empty tank at one of the gas stations.

Return the starting gas station's index if you can travel around the circuit once in the clockwise direction, otherwise return -1.

Note:

If there exists a solution, it is guaranteed to be unique.

Both input arrays are non-empty and have the same length.

Each element in the input arrays is a non-negative integer.

### Example 1:

#### **Input:**

```
gas = [1,2,3,4,5]
cost = [3,4,5,1,2]
```

#### **Output:** 3

#### **Explanation:**

Start at station 3 (index 3) and fill up with 4 unit of gas. Your tank =  $0 + 4 = 4$

Travel to station 4. Your tank =  $4 - 1 + 5 = 8$

Travel to station 0. Your tank =  $8 - 2 + 1 = 7$

Travel to station 1. Your tank =  $7 - 3 + 2 = 6$

Travel to station 2. Your tank =  $6 - 4 + 3 = 5$

Travel to station 3. The cost is 5. Your gas is

just enough to travel back to station 3.  
Therefore, return 3 as the starting index.

### Example 2:

#### **Input:**

```
gas = [2,3,4]
cost = [3,4,3]
```

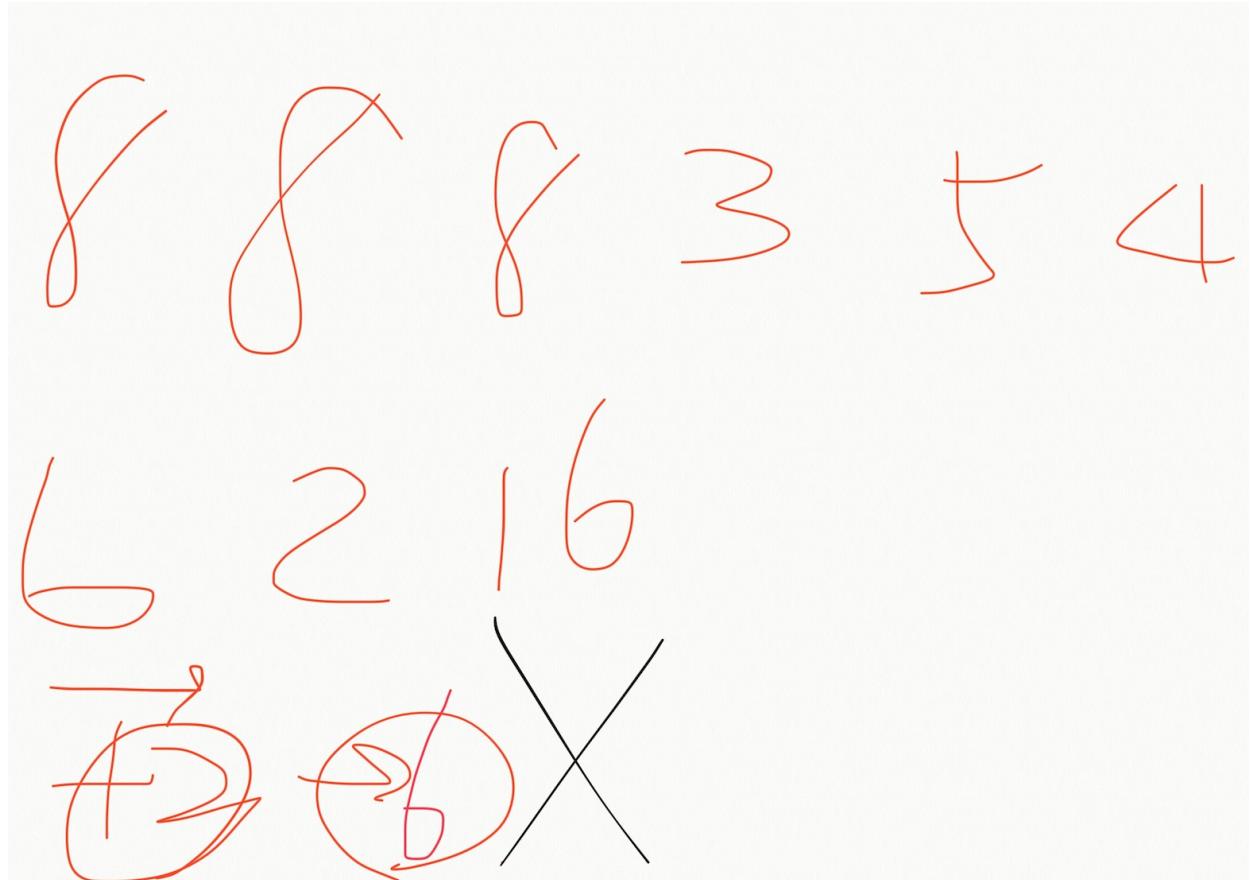
#### **Output:** -1

#### **Explanation:**

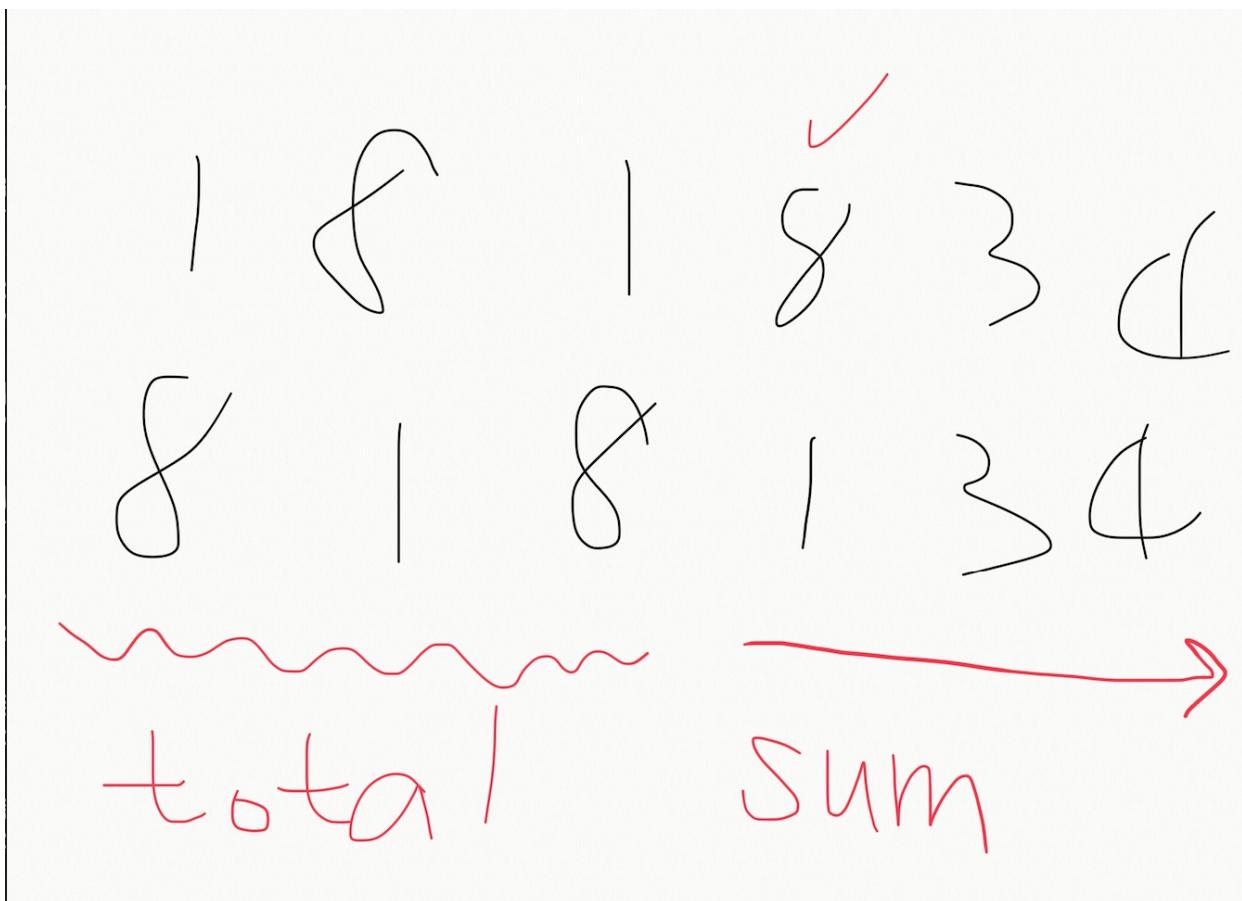
You can't start at station 0 or 1, as there is not enough gas to travel to the next station.  
Let's start at station 2 and fill up with 4 unit of gas. Your tank =  $0 + 4 = 4$   
Travel to station 0. Your tank =  $4 - 3 + 2 = 3$   
Travel to station 1. Your tank =  $3 - 3 + 3 = 3$   
You cannot travel back to station 2, as it requires 4 unit of gas but you only have 3.  
Therefore, you can't travel around the circuit once no matter where you start.

## 思路

Assumed that we walk from station 0 to 3, then blocked at 4,  
We don't have to try to start from point 1 or 2 or 3, CUZ:  
the past path gives us more gas to pass the 4, but we still fail,  
then start from 1,2,3 means we have less gas than start from 0.  
Therefore, everytime we met the failure circumstance,  
we start from the next station of the failure station.



If we start from 4 then We don't need to back to walk through 0123, we just need to make sure the total fuel minus total cost of gas is  $\geq 0$ , if it is true, then sure the circuit is true; otherwise is false.



## Code

```

class Solution {
public:
 int canCompleteCircuit(vector<int>& gas, vector<int>& cost) {
 int net = 0; // net profit n. 濕利 (or total)
 int sum = 0; // current sum of net fuel
 int start = 0; // the start of path
 for (int i=0; i< gas.size(); i++){
 net += gas[i] - cost[i];
 sum += gas[i] - cost[i];
 if (sum < 0){
 start = i+1;
 sum = 0;
 }
 }
 return (net >= 0) ? start : -1;
 }
};

```

Given an array of citations (each citation is a non-negative integer) of a researcher, write a function to compute the researcher's h-index.

According to the definition of h-index on Wikipedia: "A scientist has index  $h$  if  $h$  of his/her  $N$  papers have at least  $h$  citations each, and the other  $N - h$  papers have no more than  $h$  citations each."

### Example:

**Input:** `citations = [3,0,6,1,5]`

**Output:** 3

**Explanation:** `[3,0,6,1,5]` means the researcher has 5 papers in total and each of them had received 3, 0, 6, 1, 5 citations respectively.

Since the researcher has 3 papers with **at least 3** citations each and the remaining two with **no more than 3** citations each, her h-index is 3.

**Note:** If there are several possible values for  $h$ , the maximum one is taken as the h-index.

## 思路

按照如下方法確定某人的H指數：

- 1、將其發表的所有SCI論文按被引次數從高到低排序；
- 2、從前往後查找排序後的列表，直到某篇論文的序號大於該論文被引次數。所得序號減一即為H指數。

我也就沒多想，直接按照上面的方法寫出了代碼：

## Code

```
class Solution {
public:
 int hIndex(vector<int>& citations) {
 sort(citations.begin(), citations.end(), greater<int>());
 for (int i=0; i < citations.size(); i++) {
```

```
 if (i >= citations[i]) return i;
 }
 return citations.size();
}
};
```

---

## 275. H-Index II | 7/16

---

Given an array of citations sorted in ascending order (each citation is a non-negative integer) of a researcher, write a function to compute the researcher's h-index.

According to the definition of h-index on Wikipedia: "A scientist has index  $h$  if  $h$  of his/her  $N$  papers have at least  $h$  citations each, and the other  $N - h$  papers have no more than  $h$  citations each."

## Example:

**Input:** citations = [0,1,3,5,6]

**Output:** 3

**Explanation:** [0,1,3,5,6] means the researcher has 5 papers in total and each of them had received 0, 1, 3, 5, 6 citations respectively.

Since the researcher has 3 papers with at least 3 citations each and the remaining two with no more than 3 citations each, her h-index is 3.

## Note:

If there are several possible values for  $h$ , the maximum one is taken as the h-index.

## Follow up:

- This is a follow up problem to H-Index, where citations is now guaranteed to be sorted in ascending order.
- Could you solve it in logarithmic time complexity?

## 思路

This is the question of the #274,  
 $O(\log(n))$  implies we should use binary search to solve this.

According to 274, this question should solve by sort from greater to smaller, however we are given the opposite of it, therefore we can prefix the return with 'len - x'

## Code

```
class Solution {
```

```

public:
 int hIndex(vector<int>& citations) {
 int len = citations.size(), left = 0, right = len-1;
 while (left <= right) {
 int mid = 0.5 * (left+right);
 if (len - mid == citations[mid]) return len - mid;
 else if (len - mid > citations[mid]) left = mid + 1;
 else right = mid - 1;
 }
 return len - left;
 }
};

```

## 243. Shortest Word Distance (\$) | 7/17

---

Given a list of words and two words word1 and word2, return the shortest distance between these two words in the list.

For example,

Assume that words = ["practice", "makes", "perfect", "coding", "makes"].

Given word1 = "coding", word2 = "practice", return 3.

Given word1 = "makes", word2 = "coding", return 1.

Note:

You may assume that word1 does not equal to word2, and word1 and word2 are both in the list.

### 思路

1. Naive solution is to memorize the index of each words.
2. We can optimize our solution with only one for-loop,  
To do so, we can't memorize the index of each matching words,  
instead we should calculate the min distance when we walk through the array.

We need two value to calculate the min distance, but we can use "i" as one value.  
therefore, we just need to allocate one other variable idx.

### Code

Naive Solution:

```

class Solution {
public:
 int shortestDistance(vector<string>& words, string word1, string word2) {
 vector<int> idx1, idx2;
 int res = INT_MAX;
 for (int i=0; i < words.size(); i++){
 if (words[i] == word1) idx1.push_back(i);
 else if (words[i] == word2) idx2.push_back(i);
 }

 for (int i=0; i < idx1.size(); i++){
 for (int j=0; j < idx2.size(); j++){
 res = min(res, abs(idx1[i]-idx2[j]));
 }
 }
 return res;
 }
};

```

Optimize Solution:

```

class Solution {
public:
 int shortestDistance(vector<string>& words, string word1, string word2) {
 int idx = -1, res = INT_MAX;
 for (int i=0; i < words.size(); i++){
 if (words[i] == word1 || words[i] == word2){
 if (idx != -1 && words[i] != words[idx]){
 res = min(res, i - idx);
 }
 idx = i; // update the idx fowardly
 }
 }
 return res;
 }
};

```

## 217. Contains Duplicate | 7/18

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.

### Example 1:

**Input:** [1,2,3,1]  
**Output:** true

### Example 2:

**Input:** [1,2,3,4]  
**Output:** false

### Example 3:

**Input:** [1,1,1,3,3,4,3,2,4,2]  
**Output:** true

## 思路

Sort Solution 99% for time, 99% for space

## Code

Hash map solution:

```
class Solution {
public:
 bool containsDuplicate(vector<int>& nums) {
 unordered_map<int, int> m;
 for (int i=0; i < nums.size(); i++){
 if (m.find(nums[i]) != m.end()) return true;
 ++ m[nums[i]];
 }
 return false;
 }
};
```

Sort solution:

```
class Solution {
```

```
public:
bool containsDuplicate(vector& nums) {
if (nums.empty()) return false;
sort(nums.begin(), nums.end());
for (int i=0; i<nums.size()-1; i++){
if (nums[i] == nums[i+1]) return true;
}
return false;
}
};
```

---

## 55. Jump Game | 8/8

---

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

### Example 1:

**Input:** [2,3,1,1,4]

**Output:** true

**Explanation:** Jump 1 step from index 0 to 1, then 3 steps to the last index.

### Example 2:

**Input:** [3,2,1,0,4]

**Output:** false

**Explanation:** You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.

## 思路

1. We care about the farthest distance we can reach.
2. the goal distance is  $n-1$
3. how to determine that we can't walk further? next point's idx > max reach right now

We keep checking if we reach the goal in every iteration, we can stop looping if 1. we reach the goal, or 2. we can't move. we keep updating the farthest position in every iteration.

## Code

```
bool canJump(vector<int>& nums) {
 int n = nums.size(), reach = 0;
 for (int i=0; i < n-1; i++) {
 if (i > reach || reach >= n-1) break;
 reach = max(reach, i+nums[i]);
 }
 return reach >= n-1;
}
```

## 45. Jump Game II | 8/10

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Your goal is to reach the last index in the minimum number of jumps.

### Example:

**Input:** [2,3,1,1,4]

**Output:** 2

**Explanation:** The minimum number of jumps to reach the last index is 2.

Jump 1 step from index 0 to 1, then 3 steps to the last index.

### Note:

You can assume that you can always reach the last index.

## 思路

We should use the Greedy's opinion to solve this question.

Predict the farthest destination this step can be (if not reach the goal).

Then calculate if there is opportunity to goal in next step between the last farthest step to this farthest step.

If there is, then the solution is don't go that far in this step, shorten it then move with the goal process.

If there is not, we still can find the farthest position we can reach.

2. 3. 1. 1. 4.  
1 (1)  
1 (3) < 2 (1)  
2 (1) 3 (goal)

1. 先知道未来最多可走几步
2. 擔心步数小的反而未来可走更远?  
No worries! 步数大的会涵盖小的!

## Code

```
class Solution {
public:
```

```
int jump(vector<int>& nums) {
 int n = nums.size();
 int cur = 0, res = 0, i = 0;
 while (i < n-1){
 res++;
 int pre = cur; // setting the limitation of distance can walk
 for (; i <= pre; i++){
 cur = max(cur, i+nums[i]); // walk through, get the max distance it
 if (cur >= n-1) return res;
 }
 }
 return res; // question assume always reach the last index
};
```

---

## [Start to preparing for FB interview]

---

src 1:

[https://docs.google.com/document/d/1dIUOwN6ybeFb6PVMJdQF0QXoV6XYgchRhajNgu\\_LaRs/](https://docs.google.com/document/d/1dIUOwN6ybeFb6PVMJdQF0QXoV6XYgchRhajNgu_LaRs/)

from : <https://www.1point3acres.com/bbs/interview/facebook-software-engineer-499038.html>

重複出現: 273, 143, 253, 278, 560, 116, 117, 160

之後要重做: 34, 297, 101, 21. 23, 543, 243, 301(Hard)

---

## 273. Integer to English Words | 8/10

---

Convert a non-negative integer to its english words representation. Given input is guaranteed to be less than  $2^{31} - 1$ .

### Example 1:

**Input:** 123

**Output:** "One Hundred Twenty Three"

### Example 2:

**Input:** 12345

**Output:** "Twelve Thousand Three Hundred Forty Five"

### Example 3:

**Input:** 1234567

**Output:** "One Million Two Hundred Thirty Four Thousand Five Hundred Sixty Seven"

### Example 4:

**Input:** 1234567891

**Output:** "One Billion Two Hundred Thirty Four Million Five Hundred Sixty Seven Thousand Eight Hundred Ninety One"

Follow up:

what if we want to print the number in reverse order without reversing the number itself.  
For example, in this case : 124 will be given as input and output should be four hundred twenty one?

Constraint here is you are not allowed to reverse the number.

思路

Easy solution:

very easy!

Difficult solution:

1.  $2^{10}$  is 1000,  $2^{30}$  is  $1000^3$ , num not greater than trillion, we can used up to billion
2. 3-digit each time -> translate 3-digit
  - list the basic translation: 1~19 and 20~90
  - deal with tenth number first, then put on hundred number
  - deal with 0, 00, 000 case
  - 1. used ""
  - 2. res = t > 19 ? v2[t/10] + " " + v1[d] : v1[t]; -> WRONG; redundant space if d=0
3. check if the back has space
4. check if the input is zero

## Code

Directly & easy & fast solution:

```
class Solution {
public:
 string digits[20] = {"Zero", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Eleven", "Twelve", "Thirteen", "Fourteen", "Fifteen", "Sixteen", "Seventeen", "Eighteen", "Nineteen"};
 string tens[10] = {"Zero", "Ten", "Twenty", "Thirty", "Forty", "Fifty", "Sixty", "Seventy", "Eighty", "Ninety"};

 string int2string(int n) {
 if (n >= 1000000000) {
 return int2string(n / 1000000000) + " Billion" + int2string(n % 1000000000);
 } else if (n >= 1000000) {
 return int2string(n / 1000000) + " Million" + int2string(n % 1000000);
 } else if (n >= 1000) {
 return int2string(n / 1000) + " Thousand" + int2string(n % 1000);
 } else if (n >= 100) {
 return int2string(n / 100) + " Hundred" + int2string(n % 100);
 } else if (n >= 20) {
 return " " + tens[n / 10] + int2string(n % 10);
 } else if (n >= 1) {
 return " " + digits[n];
 } else {
 return "";
 }
 }

 string numberToWords(int num) {
 if (num == 0) {
 return "Zero";
 } else {
```

```

 string ret = int2string(num);
 return ret.substr(1, ret.length() - 1);
 }
}
};


```

Difficult strategy:

```

class Solution {
public:
 string numberToWords(int num) {
 if (num == 0) return "Zero";
 string res = transHundred(num % 1000);
 vector<string> v = {"Thousand", "Million", "Billion"};
 for (int i=0; i<3; i++){
 num /= 1000;
 res = num % 1000 ? transHundred(num % 1000) + " " + v[i] + " " + res : res;
 }
 if (res.back() == ' ') res.pop_back();
 return res;
 }

 string transHundred(int num){
 vector<string> v1 = {"", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine"};
 vector<string> v2 = {"", "", "Twenty", "Thirty", "Forty", "Fifty", "Sixty", "Seventy", "Eighty", "Ninety"};

 int h = num/100; // hundred digit: 0~9
 int t = num % 100; // ten number: 0~99
 int d = num % 10; // digit: 0~9
 string res;

 // deal with ten number
 res = t > 19 ? v2[t/10] + (d ? " " + v1[d] : "") : v1[t];
 // put hundred number up
 if (h > 0)
 res = v1[h] + " Hundred" + (t ? " "+ res : "");
 return res;
 }
};

```

Follow up:

```

class Solution {

private final String[] belowTen = new String[] {"", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine"};
private final String[] belowTwenty = new String[] {"Ten", "Eleven", "Twelve", "Thirteen", "Fourteen", "Fifteen", "Sixteen", "Seventeen", "Eighteen", "Nineteen"};
private final String[] belowHundred = new String[] {"", "Ten", "Twenty", "Thirty", "Forty", "Fifty", "Sixty", "Seventy", "Eighty", "Ninety"};

```

```

public String numberToWords(int num) {
 if(num == 0) {
 return "Zero";
 }
 return helper(num);
}
public String helper(int num) {
 String result = "";
 int length = (num+"").length();
 if(num < 10) {
 return belowTen[num];
 }
 else if(length < 3) {
 int d1 = num/10 , d2 = num%10;
 if(d2 == 1) {
 return belowTwenty[d2];
 }
 else {
 result = belowHundred[d2] + " " + belowTen[d1];
 }
 }
 else if(length == 3) {
 result = helper(num%10) + " Hundred " + helper(num/10);
 }
 else if(length == 4) {
 result = belowTen[num%10] + " Thousand " + helper(num/10);
 }
 else if(length == 5) {
 result = helper(num%100) + " Thousand " + helper(num/100);
 }
 else if(length == 6) {
 result = helper(num%1000) + " Thousand " + helper(num/1000);
 }
 else if(length == 7) {
 result = belowTen[num%10] + " Million " + helper(num/10);
 }
 else if(length == 8) {
 result = helper(num%100) + " Million " + helper(num/100);
 }
 else if(length == 9) {
 result = helper(num%1000) + " Million " + helper(num/1000);
 }
 else if(length == 10) {
 result = belowTen[num%10] + " Billion " + helper(num/10);
 }
 else if(length == 11) {
 result = helper(num%100) + " Billion " + helper(num/100);
 }
 else if(length == 12) {
 result = helper(num%1000) + " Billion " + helper(num/1000);
 }
 return result.trim();
}

```



## 257. Binary Tree Paths

Asking for iterative way in interview.

## 57. Insert Interval | 7/18

Given a set of non-overlapping intervals, insert a new interval into the intervals (merge if necessary).

You may assume that the intervals were initially sorted according to their start times.

### Example 1:

```
Input: intervals = [[1,3],[6,9]], newInterval = [2,5]
Output: [[1,5],[6,9]]
```

### Example 2:

```
Input: intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8]
Output: [[1,2],[3,10],[12,16]]
Explanation: Because the new interval [4,8] overlaps with [3,5],[6,7],[8,10].
```

## 思路

這道題讓我們在一系列非重疊的區間中插入一個新的區間，可能還需要和原有的區間合併，我們可以對給定的區間集進行一個一個的遍歷比較，那麼會有兩種情況，重疊或是不重疊，不重疊的情況最好，直接將新區間插入到對應的位置即可，重疊的情況比較複雜，有時候會有多個重疊，我們需要更新新區間的範圍以便包含所有重疊，之後將新區間加入結果 res，最後將後面的區間

再加入結果 res 即可。具體思路是，我們用一個變量 cur 來遍歷區間，如果當前 cur 區間的結束位置小於要插入的區間的起始位置的話，說明沒有重疊，則將 cur 區間加入結果 res 中，然後 cur 自增1。直到有 cur 越界或有重疊 while 循環退出，然後再用一個 while 循環處理所有重疊的區間，每次用取兩個區間起始位置的較小值，和結束位置的較大值來更新要插入的區間，然後 cur 自增1。直到 cur 越界或者沒有重疊時 while 循環退出。之後將更新好的新區間加入結果 res，然後將 cur 之後的區間再加入結果 res 中即可，

## Code

```
class Solution {
public:
 vector<vector<int>> insert(vector<vector<int>>& intervals, vector<int>& newInterval) {
 vector<vector<int>> res;
 vector<int> cur;
 int left=0, n = intervals.size();

 for (int i=0; i<n; i++){
 cur = intervals[i];
 if (newInterval[0] > cur[1]){ // the interval which don't need combination
 left++;
 res.push_back(cur);
 }
 else if (newInterval[1] < cur[0]){ // the interval which don't need combination
 continue;
 }
 else{ // combination the interval
 left++;
 newInterval[0] = min(newInterval[0], cur[0]);
 newInterval[1] = max(newInterval[1], cur[1]);
 }
 }
 res.push_back(newInterval);
 for (int i=left; i<n; i++){ // push no combination and larger interval
 res.push_back(intervals[i]);
 }
 return res;
 }
};
```

## 766. Toeplitz Matrix | 8/11

A matrix is Toeplitz if every diagonal from top-left to bottom-right has the same element.

Now given an M x N matrix, return True if and only if the matrix is Toeplitz.

### Example 1:

**Input:**

```
matrix = [
 [1,2,3,4],
 [5,1,2,3],
 [9,5,1,2]
]
```

**Output:** True**Explanation:**

In the above grid, the diagonals are:

"[9]", "[5, 5]", "[1, 1, 1]", "[2, 2, 2]", "[3, 3]", "[4]".

In each diagonal all elements are the same, so the answer is True.

### Example 2:

**Input:**

```
matrix = [
 [1,2],
 [2,2]
]
```

**Output:** False**Explanation:**

The diagonal "[1, 2]" has different elements.

Note:

matrix will be a 2D array of integers.

matrix will have a number of rows and columns in range [1, 20].

matrix[i][j] will be integers in range [0, 99].

Follow up:

What if the matrix is stored on disk, and the memory is limited such that you can only load at most one row of the matrix into the memory at once?

What if the matrix is so large that you can only load up a partial row into the memory at once?

## 思路

## Code

Directly Solution:

```
class Solution {
public:
 bool isToeplitzMatrix(vector<vector<int>>& matrix) {
 int m = matrix.size(), n = matrix[0].size();
 for (int i=0; i< m-1; i++){
 for (int j=0; j< n-1; j++){
 if (matrix[i][j] != matrix[i+1][j+1]) return false;
 }
 }
 return true;
 }
};
```

Load two row solution:

```
class Solution {
public:
 bool isToeplitzMatrix(vector<vector<int>>& matrix) {
 int m = matrix.size();
 for(int i = 0; i < m-1; ++i){
 vector<int> m1 = matrix[i];
 vector<int> m2 = matrix[i+1];
 m1.pop_back();
 m2.erase(m2.begin());
 if(m1!=m2) return false;
 }
 return true;
 }
};
```

---

Given a singly linked list L:  $L_0 \rightarrow L_1 \rightarrow \dots \rightarrow L_{n-1} \rightarrow L_n$ ,  
reorder it to:  $L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow L_2 \rightarrow L_{n-2} \rightarrow \dots$

You may not modify the values in the list's nodes, only nodes itself may be changed.

### Example 1:

Given  $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ , reorder it to  $1 \rightarrow 4 \rightarrow 2 \rightarrow 3$ .

### Example 2:

Given  $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$ , reorder it to  $1 \rightarrow 5 \rightarrow 2 \rightarrow 4 \rightarrow 3$ .

## 思路

The directly think of way is divided into three steps:

- find out the mid
- reverse the nodes after mid
- insert to the front one intervally

The nodes after the mid need to be reversely add to the front list between original nodes

We can use the stack to imprement the "reverse" idea (First-in-last-out)

We can know exactly how much nodes need to be reversely added by getting the middle index of the stack.

## Code

```
class Solution {
public:
 void reorderList(ListNode* head) {
 if (!head || !head->next || !head->next->next) return; // testcase: 1 => 1
 // use the slow fast pointer to find the mid point
 ListNode* slow = head;
 ListNode* fast = head;
 while (fast->next && fast->next->next){
 slow = slow->next;
 fast = fast->next->next;
```

```

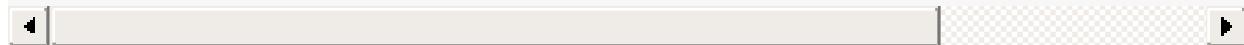
 }

 // reverse the list after mid
 ListNode* mid = slow->next;
 slow->next = NULL; // cut off the post-mid list, and add an NULL to list head
 // Now, we have two list: a. head, b. mid
 // Then, we're going to reverse the list mid

 ListNode* insertingList = NULL; // need a new head for reversed list
 while (mid){
 ListNode* temp = mid->next;
 mid->next = insertingList;
 insertingList = mid;
 mid = temp;
 }
 // insert insertingList intervally to the front list(which is head list)
 while (insertingList && head){
 ListNode* next = head->next; // record the inserted node
 head->next = insertingList; // replace with inserted node
 insertingList = insertingList->next; // iterating the next inserting node
 head->next->next = next; // re-connected the head list
 head = next;// iterating the next inserted point of head
 }
}

};


```



```

class Solution {
public:
 void reorderList(ListNode* head) {
 stack<ListNode*> s;
 ListNode* cur = head;
 while (cur){
 s.push(cur);
 cur = cur->next;
 }
 cur = head; // used to point the inserting pos
 int needReverse = (s.size()-1)/2;
 while (needReverse-- > 0){
 ListNode* top = s.top(); s.pop();
 ListNode* next = cur->next; // record the next of inserting point
 cur->next = top; // insert to the point
 top->next = next; // rebuild the link
 cur = next; // move to next inserting point
 }
 s.top()->next = NULL; // required this or it'll form circle
 }
};


```



## 67. Add Binary | 8/13

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

The input strings are both non-empty and contains only characters 1 or 0.

### Example 1:

**Input:** a = "11", b = "1"

**Output:** "100"

### Example 2:

**Input:** a = "1010", b = "1011"

**Output:** "10101"

## 思路

1. should be the same Length
2. insert from front
3. deal with the carry

## Code

```
class Solution {
public:
 string addBinary(string a, string b) {
 string res;
 int na = a.length();
 int nb = b.length();
 int n = max(na, nb);
 if (na > nb){
 for (int i=0; i < na - nb; i++) b.insert(b.begin(), '0');
 }
 else if (nb > na){
 for (int i=0; i < nb - na; i++) a.insert(a.begin(), '0');
 }
 bool carry = false;
 for (int i = n-1; i >= 0; i--){
 int sum = 0;
```

```

 if (carry){
 sum = ((a[i]-'0') + (b[i]-'0') + 1);
 }
 else{
 sum = ((a[i]-'0') + (b[i]-'0'));
 }

 if (sum == 0){
 res.insert(res.begin(), '0');
 carry = false;
 }
 else if (sum == 1){
 res.insert(res.begin(), '1');
 carry = false;
 }
 else if (sum == 2){
 res.insert(res.begin(), '0');
 carry = true;
 }
 else if (sum == 3){
 res.insert(res.begin(), '1');
 carry = true;
 }
 }
 if (carry) res.insert(res.begin(), '1');
 return res;
}
};

```

---

## 76. Minimum Window Substring | 8/15

---

Given a string S and a string T, find the minimum window in S which will contain all the characters in T in complexity O(n).

### Example:

**Input:** S = "ADOBECODEBANC", T = "ABC"

**Output:** "BANC"

### Note:

- If there is no such window in S that covers all characters in T, return the empty string "".
- If there is such window, you are guaranteed that there will always be only one unique minimum window in S.

## 思路

目前在腦子一片漿糊的情況下，我們還是從簡單的例子來分析吧，題目例子中的S有點長，換個短的 S = "ADBANC", T = "ABC"，那麼肉眼遍歷一遍S唄，首先第一個是A，嗯很好，T中有，第二個是D，T中沒有，不理它，第三個是B，嗯很好，T中有，第四個又是A，多了一個，禮多人不怪嘛，收下啦，第五個是N，一邊涼快去，第六個終於是C了，那麼貌似好像需要整個S串，其實不然，注意之前有多一個A，就算去掉第一個A，也沒事，因為第四個A可以代替之，第二個D也可以去掉，因為不在T串中，第三個B就不能再去掉了，不然就沒有B了。所以最終的答案就"BANC"了。通過上面的描述，你有沒有發現一個有趣的現象，先擴展，再收縮，就好像一個窗口一樣，先擴大右邊界，然後再收縮左邊界，上面的例子中右邊界無法擴大了後才開始收縮左邊界，實際上對於複雜的例子，有可能是擴大右邊界，然後縮小一下左邊界，然後再擴大右邊界等等。這就很像一個不停滑動的窗口了，這就是大名鼎鼎的滑動窗口 Sliding Window 了，簡直是神器啊，能解很多子串，子數組，子序列等等的問題，是必須要熟練掌握的啊！

- 先掃瞄一遍T，把對應的字符及其出現的次數存到 HashMap 中。
- 然後開始遍歷S，就把遍歷到的字母對應的 HashMap 中的 value 減一，如果減1後仍大於等於0，cnt 自增1。
- 如果 cnt 等於T串長度時，開始循環，紀錄一個字串並更新最小字串值。然後將子窗口的左邊界向右移，如果某個移除掉的字母是T串中不可缺少的字母，那麼 cnt 自減1，表示此時T串並沒有完全匹配。

{  
Add: mines  
Delete: plus

ABC

ADOBE CODEBANC



Map的3種狀態：

A : 1, 0, 1, 0

a. 大於0 : window缺少

B : 1, 0, -1, 0

b. 等於0 : window有1

C : 1, 0, 1, 0

c. 小於0 : window有1+個

D -1

E -1.

\* 非目標 str 的 character,永遠不會 > 0  
因為在移動視窗中的加減都是固定的.  
外loop走過,做減. 內loop收縮做加.

Code

```
/*
The sliding window strategy to solve:
```

```

Current size of the window: right bound - left bound + 1
The min size of the string: minStr
The counter to make sure our window contain the string t: cnt
The current left bound: left
The current right bound: i
*/
class Solution {
public:
 string minWindow(string s, string t) {
 unordered_map<char, int> windowMap;
 string res = "";
 int minStr = INT_MAX;
 int cnt = 0;
 int left = 0;

 for (char c : t) windowMap[c]++;
 for (int i=0; i < s.size(); i++){
 // Walk through the string s, broaden &
 // broaden our right-side
 if (--windowMap[s[i]] >= 0) cnt++;
 // update the current map and current window

 // if our current window include all the character we require
 while (cnt == t.size()){
 // record the current string if it is shorter than past one
 if (minStr > (i - left + 1)){
 minStr = i-left+1;
 res = s.substr(left, minStr);
 }

 // start to shorten from left
 if (++windowMap[s[left]] > 0) cnt--;
 left++;
 }
 }
 return res;
 }
};

```

## 349. Intersection of Two Arrays | 8/15

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

### Example 1:

**Input:** nums1 = [1,2,2,1], nums2 = [2,2]  
**Output:** [2]

### Example 2:

**Input:** nums1 = [4,9,5], nums2 = [9,4,9,8,4]  
**Output:** [9,4]

### Note:

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

## 思路

用Hashtable去操作元素是否存在，以及各數為何  
新增進res後可以將映射值還原成0，來避免下次的新增

## Code

```
class Solution {
public:
 vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {
 vector<int> res;
 unordered_map<int, int> nums1_map;
 for (int i: nums1) nums1_map[i]++;
 for (int i=0; i < nums2.size(); i++){
 if (-- nums1_map[nums2[i]] >= 0){
 res.push_back(nums2[i]);
 nums1_map[nums2[i]] = 0; // eliminate redundant element
 }
 }
 return res;
 }
};
```

unordered\_set solution

```
class Solution {
public:
 vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {
 unordered_set<int> m(nums1.begin(), nums1.end());
 vector<int> res;
 for (auto a : nums2)
 if (m.count(a)) {
 res.push_back(a);
 m.erase(a);
 }
 return res;
 }
};
```

## 15. 3Sum | 8/18

Given an array `nums` of  $n$  integers, are there elements  $a$ ,  $b$ ,  $c$  in `nums` such that  $a + b + c = 0$ ?  
Find all unique triplets in the array which gives the sum of zero.

Note:

The solution set must not contain duplicate triplets.

### Example:

Given array `nums` = `[-1, 0, 1, 2, -1, -4]`,

A solution set is:

```
[
 [-1, 0, 1],
 [-1, -1, 2]
]
```

## 思路

對原數組進行排序，然後開始遍歷排序後的數組，這裡注意不是遍歷到最後一個停止，而是到倒

數第三個就可以了。這裡可以先做個剪枝優化，就是當遍歷到正數的時候就 break，為什麼呢，因為數組現在是有序的了，如果第一個要 fix 的數就是正數了，則後面的數字就都是正數，就永遠不會出現和為0的情況了。然後還要加上重複就跳過的處理，處理方法是從第二個數開始，如果和前面的數字相等，就跳過，因為不想把相同的數字fix兩次。對於遍歷到的數，用0減去這個 fix 的數得到一個 target，然後只需要再之後找到兩個數之和等於 target 即可。用兩個指針分別指向 fix 數字之後開始的數組首尾兩個數，如果兩個數和正好為 target，則將這兩個數和 fix 的數一起存入結果中。然後就是跳過重複數字的步驟了，兩個指針都需要檢測重複數字。如果兩數之和小於 target，則將左邊那個指針i右移一位，使得指向的數字增大一些。同理，如果兩數之和大於 target，則將右邊那個指針j左移一位，使得指向的數字減小一些

## Code

```
class Solution {
public:
 vector<vector<int>> threeSum(vector<int>& nums) {
 vector<vector<int>> res;
 sort(nums.begin(), nums.end());
 if (nums.size() < 3 || nums.back() < 0 || nums.front() > 0) return res;

 for (int i=0; i < nums.size() - 2; i++){
 if (nums[i] > 0) break;
 if (i-1 >= 0 && nums[i] == nums[i-1]) continue;
 int target = 0 - nums[i]; // fix one nums and calculate the target of o

 // use two pointer to converge and find if meet the target
 int left = i+1;
 int right = nums.size() - 1;
 while (left < right){
 int sum = nums[left] + nums[right];
 if (sum < target) left += 1;
 else if (sum > target) right -= 1;
 else {
 vector<int> comb;
 comb.push_back(nums[left]);
 comb.push_back(nums[i]);
 comb.push_back(nums[right]);
 res.push_back(comb);

 // 思考：為什麼可以只在裡面做檢查就好
 while (left < right && nums[left] == nums[left+1]) left++;
 while (left < right && nums[right] == nums[right-1]) right--;
 left++;
 right--;
 } // end of else
 } // end of two pointer
 } // end of for
 return res;
 }
};
```



## 253. Meeting Rooms II (\$) | 8/18

Given an array of meeting time intervals consisting of start and end times  $[[s_1, e_1], [s_2, e_2], \dots]$  ( $s_i < e_i$ ), find the minimum number of conference rooms required.

### Example

#### Example1

Input: intervals = [(0,30),(5,10),(15,20)]

Output: 2

Explanation:

We need two meeting rooms

room1: (0,30)

room2: (5,10),(15,20)

#### Example2

Input: intervals = [(2,7)]

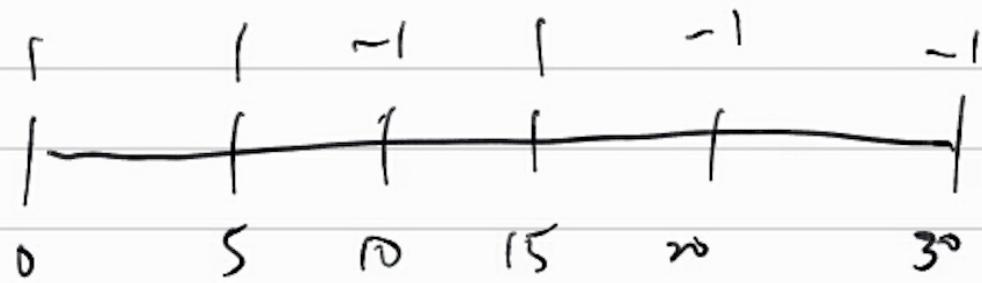
Output: 1

Explanation:

Only need one meeting room

### 思路

Note how to use Map to make  $O(N^2)$  solution to N!  
with same solving idea.



## Code

$O(N^2)$  Solution:

```
/*
 * Definition of Interval:
 * class Interval {
 * int start, end;
 * Interval(int start, int end) {
 * this->start = start;
 * this->end = end;
 * }
 * }
 */
class Solution {
public:

 int minMeetingRooms(vector<Interval> &intervals) {
 if (intervals.size() <= 1) return intervals.size();
 int n = intervals.size();
 int max_unit = INT_MIN;

 int res = INT_MIN;
 // find Max time unit
 for (int i=0; i<n; i++){
 max_unit = max(max_unit, intervals[i].end);
 }
 int timeunit[max_unit] = {0};
 // find Max number in the unit while go through each interval
 for (int i=0; i<n; i++){
 for (int j=intervals[i].start; j<max_unit; j++)
 timeunit[j]++;
 }
 int max_timeunit = 0;
 for (int i=0; i<max_unit; i++)
 max_timeunit = max(max_timeunit, timeunit[i]);
 return max_timeunit;
 }
}
```

```

 for (int j=intervals[i].start; j < intervals[i].end; j++){
 timeunit[j]++;
 res = max(res, timeunit[j]);
 }
 }
 return res;
}
};

```

O(n) solution:

```

class Solution {
public:

 int minMeetingRooms(vector<Interval> &intervals) {
 if (intervals.size() <= 1) return intervals.size();
 map<int, int> time_unit; // will auto sort
 for (auto i: intervals){
 time_unit[i.start]++;
 time_unit[i.end]--;
 }

 int openedRoom = 0;
 int res = 0;
 for (auto t : time_unit){
 res = max(res, openedRoom += t.second);
 }
 return res;
 }
};

```

## 278. First Bad Version | 8/18

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.

## Example:

Given  $n = 5$ , and  $\text{version} = 4$  is the first bad version.

```
call isBadVersion(3) -> false
call isBadVersion(5) -> true
call isBadVersion(4) -> true
```

Then 4 is the first bad version.

## 思路

Use Binary search to solve.

if mid is bad, search its left. otherwise, search its right.

0 and n should be included.

## Code

```
// Forward declaration of isBadVersion API.
bool isBadVersion(int version);

class Solution {
public:
 int firstBadVersion(int n) {
 int left = 0, right = n;
 while (left < right) {
 int mid = left + (right-left)/2;
 if (isBadVersion(mid)) {
 if (mid > 0 && !isBadVersion(mid-1)) return mid;
 else right = mid;
 }
 else {
 if (mid < n && isBadVersion(mid+1)) return mid+1;
 else left = mid;
 }
 }
 return left;
 }
};
```

## 263. Graph Valid Tree | 8/19

---

Given  $n$  nodes labeled from 0 to  $n - 1$  and a list of undirected edges (each edge is a pair of nodes), write a function to check whether these edges make up a valid tree.

For example:

Given  $n = 5$  and edges =  $[[0, 1], [0, 2], [0, 3], [1, 4]]$ , return true.

Given  $n = 5$  and edges =  $[[0, 1], [1, 2], [2, 3], [1, 3], [1, 4]]$ , return false.

Hint:

Given  $n = 5$  and edges =  $[[0, 1], [1, 2], [3, 4]]$ , what should your return? Is this case a valid tree?  
According to the definition of tree on Wikipedia: “a tree is an undirected graph in which any two vertices are connected by exactly one path. In other words, any connected graph without simple cycles is a tree.”

Note: you can assume that no duplicate edges will appear in edges. Since all edges are undirected,  $[0, 1]$  is the same as  $[1, 0]$  and thus will not appear together in edges.

### 思路

find out how the false happened by graph.

$[0,1], [1,2], [2,3], [1,3] \sqsubseteq [1,4] \Rightarrow \text{false.}$

	$(0,1)$	$(0,2)$	$(0,3)$		
$(0,0)$	0	1	2	3	4
$(1,1)$	0	✓			
$(2,1)$	1	✓		✓	✓
$(3,1)$	2		✓	✓	
3					
4					

cycle  
false

Input  $(0,1) \rightarrow (1,2) \Leftarrow \text{true.}$

|0|1|2|3|

0||T||

1|||T|T|\_ conflict!

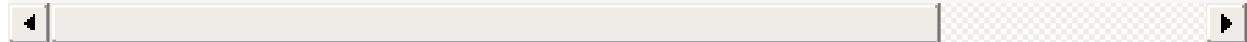
2|||T| here !!

3||||\_

Note: "HOW to initiate the 2-D array!" => use memset(arr, value, n\*sizeof(arr))

## Code

```
bool validTree(int n, vector<vector<int>> &edges) {
 if (n > 1 && edges.empty()) return false;
 bool check[n+1][n+1];
 memset(check, false, n+1*n+1*sizeof(check));
 for (int i=0; i<edges.size(); i++){
 int first = edges[i][0] + 1;
 int second = edges[i][1] + 1;
 if (first > second) swap(first,second);
 if ((first+1< n+1 && check[first+1][second]) || (first -1 > 1 && check[first-1][second]))
 return false;
 }
 check[first][second] = true;
}
return true;
}
```



## 類似263

给一个TreeNode[], 检查是否能形成一颗无cycle的树，可以的话返回root，否则返回null

我用的hashmap存了每个的parent，面完了回想一下不需要 hashmap，hashset就可以。

第二题我是这么想的，把node array中的所有node过一遍，current node的两个child如果不是null的话放在hashset里面，

iterate through的过程中如果有child已经在hashset中，证明这个child有两个parent返回null，  
node array过完一遍后检查hashset，因为node[]包含的是所有node，检查有哪些没有在hashset  
里面（没有parent的node），

如果只有一个，那这个就是root

如果一个都没有，那就是有cycle

如果多于1个，就是有好几个root，不是single tree

## 791. Custom Sort String | 8/19

S and T are strings composed of lowercase letters. In S, no letter occurs more than once.

S was sorted in some custom order previously. We want to permute the characters of T so that they match the order that S was sorted. More specifically, if x occurs before y in S, then x should occur before y in the returned string.

Return any permutation of T (as a string) that satisfies this property.

**Example :**

**Input:**

S = "cba"

T = "abcd"

**Output:** "cbad"

**Explanation:**

"a", "b", "c" appear in S, so the order of "a", "b", "c" should be "c", "b", and "a".

Since "d" does not appear in S, it can be at any position in T. "dcba", "cdba", "cbda" are also valid outputs.

## 思路

Need to know how many char in a string -> use unordered\_map;  
technique here: string(int n, char c) -> string s = string(2,'A') => "AA"

Clarify your concept: unordered\_map no auto sort for sure

PLUS: NO guarantee your input order is the output order EITHER!

## Code

```
class Solution {
public:
 string customSortString(string S, string T) {
 string res = "";
 unordered_map<char, int> m;
 for (char t: T) m[t]++;
 // record the count of c in T
 for (char s: S){
 res += string(m[s], s);
 m[s] = 0;
```

```
 }
 for (auto a: m){
 res += string(a.second, a.first);
 }
 return res;
}
};
```

---

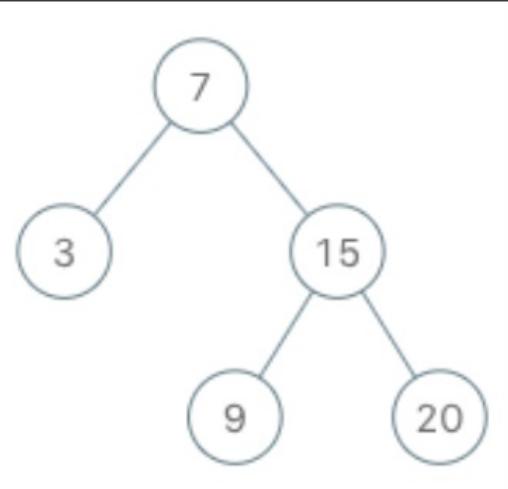
## 173. Binary Search Tree Iterator | 8/21

---

Implement an iterator over a binary search tree (BST). Your iterator will be initialized with the root node of a BST.

Calling next() will return the next smallest number in the BST.

### Example:



```
BSTIterator iterator = new BSTIterator(root);
iterator.next(); // return 3
iterator.next(); // return 7
iterator.hasNext(); // return true
iterator.next(); // return 9
iterator.hasNext(); // return true
iterator.next(); // return 15
iterator.hasNext(); // return true
iterator.next(); // return 20
iterator.hasNext(); // return false
```

### Note:

- `next()` and `hasNext()` should run in average  $O(1)$  time and uses  $O(h)$  memory, where  $h$  is the height of the tree.
- You may assume that `next()` call will always be valid, that is, there will be at least a next smallest number in the BST when `next()` is called.

思路

這道題主要就是考二叉樹的中序遍歷的非遞歸形式，需要額外定義一個棧來輔助，二叉搜索樹的建樹規則就是左<根<右，用中序遍歷即可從小到大取出所有節點。

Way 1, 參考 <https://leetcode.com/problems/binary-tree-inorder-traversal/> 的做法

Way 2, 更加快速

## Code

```
class BSTIterator {
private:
 TreeNode* cur;
 stack<TreeNode*> sta;

public:
 BSTIterator(TreeNode* root) {
 cur = root;
 }

 // return the next smallest number
 int next() {
 while (cur){
 sta.push(cur);
 cur = cur->left;
 }
 cur = sta.top(); sta.pop();
 int value = cur->val;
 cur = cur->right;
 return value;
 }

 // return whether we have a next smallest number
 bool hasNext() {
 return cur || !sta.empty();
 }
};
```

Better way:

```
class BSTIterator {
private:
 stack<TreeNode*> sta;

public:
 BSTIterator(TreeNode* root) {
 findLeft(root);
 }

 // return the next smallest number
```

```

int next() {
 TreeNode* top = sta.top(); sta.pop();
 findLeft(top->right);

 return top->val;
}

// return whether we have a next smallest number
bool hasNext() {
 if (sta.empty()) return false;
 else return true;
}

// GO EXTREMELY LEFT!
void findLeft(TreeNode* root){
 while (root){
 sta.push(root);
 root = root->left;
 }
}

```

---

## 101 Symmetric tree

---

follow up: use iterative to do

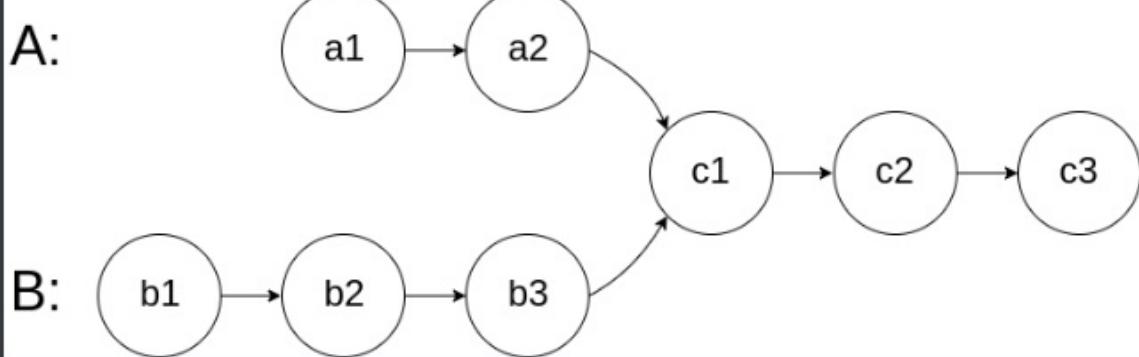
---

## 160. Intersection of Two Linked Lists | 8/21

---

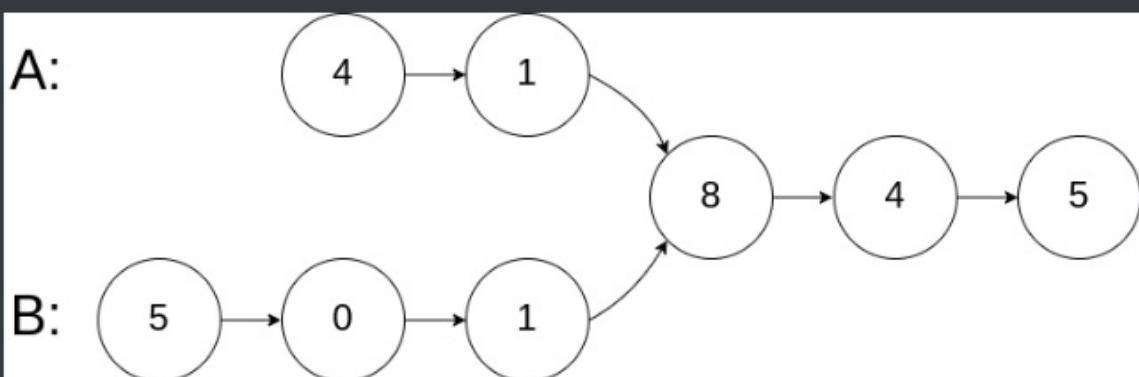
Write a program to find the node at which the intersection of two singly linked lists begins.

For example, the following two linked lists:



begin to intersect at node c1.

### Example 1:

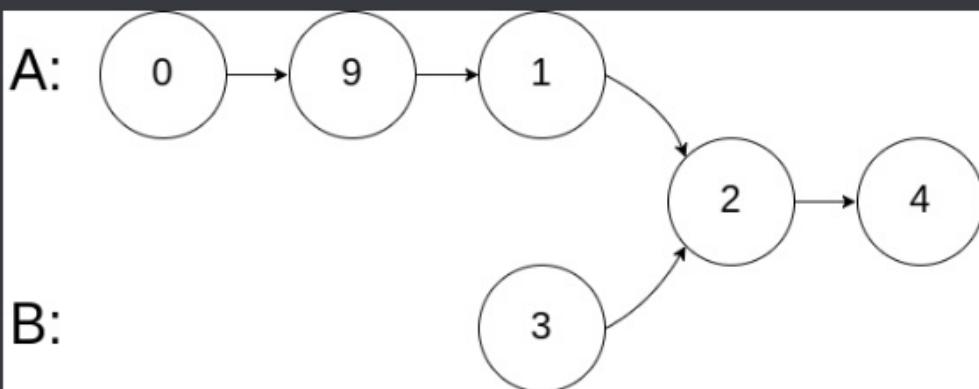


**Input:** intersectVal = 8, listA = [4,1,8,4,5],  
listB = [5,0,1,8,4,5], skipA = 2, skipB = 3

**Output:** Reference of the node with value = 8

**Input Explanation:** The intersected node's value is 8 (note that this must not be 0 if the two lists intersect). From the head of A, it reads as [4,1,8,4,5]. From the head of B, it reads as [5,0,1,8,4,5]. There are 2 nodes before the intersected node in A; There are 3 nodes before the intersected node in B.

## Example 2:

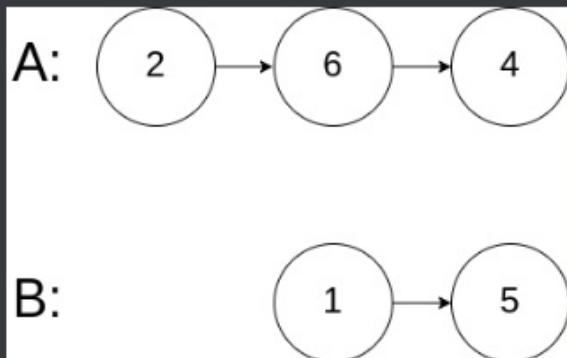


**Input:** intersectVal = 2, listA = [0,9,1,2,4],  
listB = [3,2,4], skipA = 3, skipB = 1

**Output:** Reference of the node with value = 2

**Input Explanation:** The intersected node's value is 2 (note that this must not be 0 if the two lists intersect). From the head of A, it reads as [0,9,1,2,4]. From the head of B, it reads as [3,2,4]. There are 3 nodes before the intersected node in A; There are 1 node before the intersected node in B.

### Example 3:



**Input:** intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2

**Output:** null

**Input Explanation:** From the head of A, it reads as [2,6,4]. From the head of B, it reads as [1,5]. Since the two lists do not intersect, intersectVal must be 0, while skipA and skipB can be arbitrary values.

**Explanation:** The two lists do not intersect, so return null.

### 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.

# 技巧

兩個list不一樣是要讓其中一個先走的做法：

```
for i in range(longerlist.length):
 if i >= longerlist.length - shorterlist.length:
 # comparison here
 else:
 longerlist = longerlist.next
```

# 思路

讓兩個節點從同樣剩餘長度的節點再開始比較即可

從例子一中可以看出來，相同的value的節點並不一定是相同的節點

需要考慮題目非input的skipA, skipB

# Code

```
class Solution {
public:
 ListNode* getIntersectionNode(ListNode* headA, ListNode* headB) {
 int lengthA = getLength(headA);
 int lengthB = getLength(headB);

 if (lengthA > lengthB) return findSame(headA, lengthA, headB, lengthB);
 else return findSame(headB, lengthB, headA, lengthA);

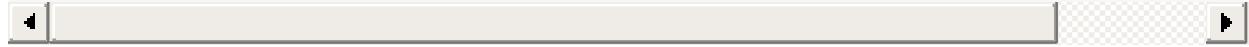
 }

 ListNode* findSame(ListNode* longer, int longerint, ListNode* shorter, int shorterint) {
 for (int i=0; i < longerint; i++){
 if (i >= (longerint-shorterint)){
 // start comparing here
 if (!longer || !shorter) return NULL;
 if (longer == shorter) return longer;
 else{
 longer = longer->next;
 shorter = shorter->next;
 }
 }
 else{
 // let longer iterate first
 if (longer)
 longer = longer->next;
 }
 }
 return NULL;
}
```

```

 }
 int getLength(ListNode* root){
 int count = 0;
 while (root){
 count++;
 root = root->next;
 }
 return count;
 }
};

```



Python

```

class Solution(object):
 def getIntersectionNode(self, headA, headB):
 """
 :type head1, head2: ListNode
 :rtype: ListNode
 """

 lengthA = self.getLength(headA)
 lengthB = self.getLength(headB)

 if lengthA >= lengthB:
 return self.getInteraction(headA, headB, lengthA, lengthB)
 else:
 return self.getInteraction(headB, headA, lengthB, lengthA)

 def getInteraction(self, longer, shorter, longl, shortl):
 for i in range(longl):
 if i >= longl-shortl:
 if not longer or not shorter:
 return None
 if longer == shorter:
 return longer
 else:
 longer = longer.next
 shorter= shorter.next
 else:
 longer = longer.next
 return None

 def getLength(self, node):
 count = 0
 while node:
 count += 1
 node = node.next
 return count

```

# 116. Populating Next Right Pointers in Each Node |

## 8/24

You are given a perfect binary tree where all leaves are on the same level, and every parent has two children. The binary tree has the following definition:

```
struct Node {
 int val;
 Node* left;
 Node* right;
 Node* next;
}
```

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.

Initially, all next pointers are set to NULL.

### Example:

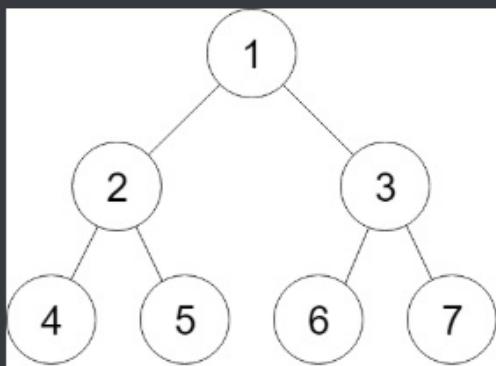


Figure A

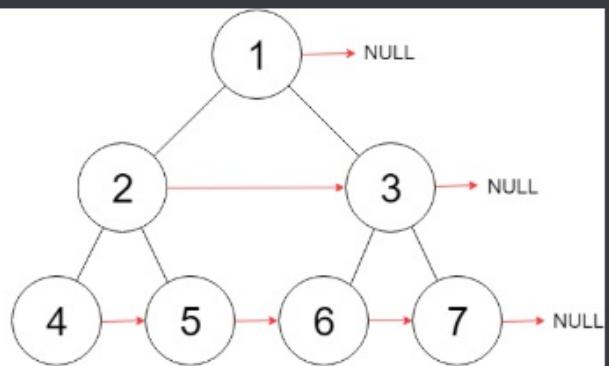


Figure B

```
Input: {"$id":"1","left":{"$id":"2","left":{"$id":"3","left":null,"next":null,"right":null,"val":3,"next":null}, "right":{"$id":"4","left":null,"next":null,"right":null,"val":4,"next":null}, "next":null}, "right":{"$id":"5","left":{"$id":"6","left":null,"next":null,"right":null,"val":6,"next":null}, "right":{"$id":"7","left":null,"next":null,"right":null,"val":7,"next":null}}
```

```
Output: {"$id":"1","left":{"$id":"2","left":{"$id":"3","left":null,"next":null}, "right":{"$id":"4","left":null,"next":null,"right":null,"val":4,"next":null}, "next":null}, "right":{"$id":"5","left":{"$id":"6","left":null,"next":null,"right":null,"val":6,"next":null}, "right":{"$id":"7","left":null,"next":null,"right":null,"val":7,"next":null}}
```

```
{"$id":"4","left":null,"next":
{"$id":"5","left":null,"next":
{"$id":"6","left":null,"next":null,"right":null,"val":
{"$id":"7","left":
{"$ref":"5"}, "next":null,"right":
{"$ref":"6"}, "val":3}, "right":
{"$ref":"4"}, "val":2}, "next":null,"right":
{"$ref":"7"}, "val":1}
```

**Explanation:** Given the above perfect binary tree (Figure A), your function should populate each next pointer to point to its next right node, just like in Figure B.

### Note:

- You may only use constant extra space.
- Recursive approach is fine, implicit stack space does not count as extra space for this problem.

## 思路

完全二叉樹，所以若節點的左子結點存在的話，其右子節點必定存在，所以左子結點的next指針可以直接指向其右子節點，對於其右子節點的處理方法是，判斷其父節點的next是否為空，若不為空，則指向其next指針指向的節點的左子結點，若為空則指向NULL

p.s. level order should use queue, instead of stack

## Code

```
class Solution {
public:
 Node* connect(Node* root) {
 if (!root) return NULL;
```

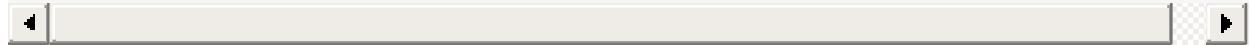
```

queue<Node*> que;
que.push(root);

while (!que.empty()){
 int levelsize = que.size();
 for (int i=0; i<levelsized; i++){
 Node* cur = que.front(); que.pop();
 if (i < levelsized-1){
 cur -> next = que.front(); // current node point the next node
 }
 else{
 cur -> next = NULL;
 }

 // update child node
 if (cur->left) que.push(cur->left);
 if (cur->right) que.push(cur->right);
 }
}
return root;
};


```



Solve without queue:

```

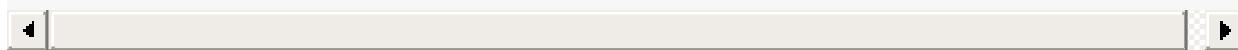
/*
use level->left to iterate the level
to build the connection with the view of parent (seen from top to down)
*/
class Solution {
public:
 Node* connect(Node* root) {
 if (!root) return NULL;
 Node* level = root;
 Node* parent = NULL;
 while (level->left){
 parent = level;
 while (parent){
 if (parent->left){
 parent->left->next = parent->right;
 }
 if (parent->next){
 parent->right->next = parent->next->left;
 }
 parent = parent->next;
 }
 level = level->left;
 }
 return root;
 };
};


```



Recursive way:

```
class Solution {
public:
 Node* connect(Node* root) {
 if (!root) return NULL;
 if (root->left) root->left->next = root->right;
 if (root->right) root->right->next = root->next? root->next->left : NULL;
 connect(root->left);
 connect(root->right);
 return root;
 }
};
```



## 117. Populating Next Right Pointers in Each Node II | 8/24

Given a binary tree

```
struct Node {
 int val;
 Node* left;
 Node* right;
 Node* next;
}
```

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.

Initially, all next pointers are set to NULL.

### Example:

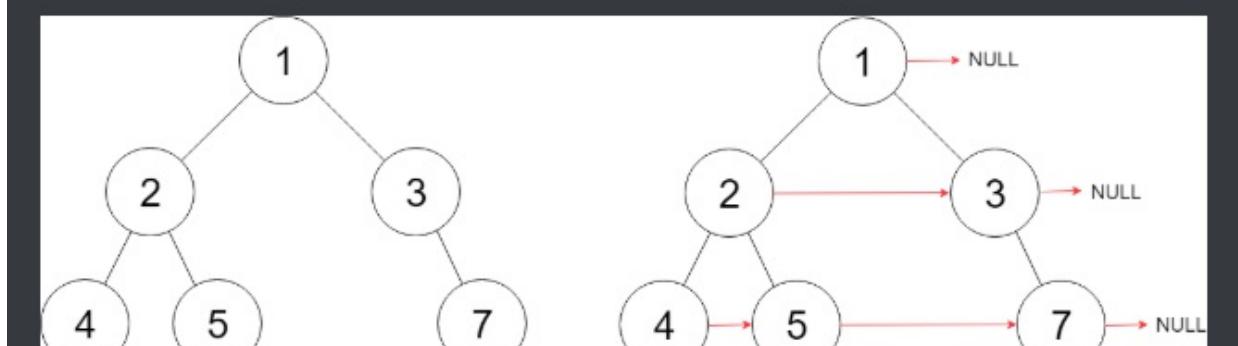


Figure A

Figure B

**Input:** {"\$id": "1", "left": {"\$id": "2", "left": {"\$id": "3", "left": null, "next": null, "right": null, "val": 1}, "right": {"\$id": "4", "left": null, "next": null, "right": null, "val": 2}, "val": 3}, "right": {"\$id": "5", "left": null, "next": null, "right": {"\$id": "6", "left": null, "next": null, "right": null, "val": 4}}, "val": 5}

**Output:** {"\$id": "1", "left": {"\$id": "2", "left": {"\$id": "3", "left": null, "next": {"\$id": "4", "left": null, "next": {"\$id": "5", "left": null, "next": {"\$id": "6", "left": null, "next": null, "right": {"\$ref": "5"}, "val": 3}, "right": {"\$ref": "4"}, "val": 2}, "val": 1}, "val": 4}, "right": {"\$id": "5", "left": null, "next": null, "right": {"\$ref": "6"}, "val": 5}}

**Explanation:** Given the above binary tree (Figure A), your function should populate each next pointer to point to its next right node, just like in Figure B.

#### Note:

- You may only use constant extra space.
- Recursive approach is fine, implicit stack space does not count as extra space for this problem.

## 思路

Non-recursive的方法很好解，並且是116&117的通解

要符合題目要求的constant space，就不能用queue  
要用dummy node的概念，但我看不懂QAQ  
詳情參考 <https://www.cnblogs.com/grandyang/p/4290148.html>

## Code

```
class Solution {
public:
 Node* connect(Node* root) {
 if (!root) return NULL;
 queue<Node*> que;
 que.push(root);

 while (!que.empty()){
 int levelnodes = que.size();
 for (int i=0; i<levelnodes; i++){
 Node* front = que.front(); que.pop();
 if (i < levelnodes-1){
 front->next = que.front();
 }
 else {
 front->next = NULL;
 }

 // Update
 if (front->left) que.push(front->left);
 if (front->right) que.push(front->right);
 }
 }

 return root;
 };
};
```

---

## 621. Task Scheduler(unsolved follow up) | 8/26

---

Given a char array representing tasks CPU need to do. It contains capital letters A to Z where different letters represent different tasks. Tasks could be done without original order. Each task could be done in one interval. For each interval, CPU could finish one task or just be idle.

However, there is a non-negative cooling interval n that means between two same tasks, there must be at least n intervals that CPU are doing different tasks or just be idle.

You need to return the least number of intervals the CPU will take to finish all the given tasks.

# 思路

1.

[AAABBB]

A:3, B:3 Unordered map?

Need sorting => vector

[AAABBBC] n = 2

AB-AB-AB =>  $(2+1)^*(3-1) + 2 = 8$

"AB-" \* "2個AB-" + AB

[AAABBBC] n = 3

AB--AB--AB =>  $(3+1)^*(3-1)+2 = 10$

[AAAAABBBC], n = 3

$4 * 4 + 1 = 15$

Equation:  $(n+1)^*(\max\_count - 1) + (\text{how\_many\_max\_exist})$

=>  $\text{how\_many\_char\_in\_interval} * \text{how\_many\_interval} + \text{rest\_char}$

edge case:

[AAABBB] n = 0

Equation:  $1^*2+2 = 4$

- Follow up: return the string
  - <https://www.1point3acres.com/bbs/thread-492747-2-1.html>

## Code

```
class Solution {
public:
 int leastInterval(vector<char>& tasks, int n) {
 // count the char
 vector<int> count(26, 0);
 for (char task: tasks){
```

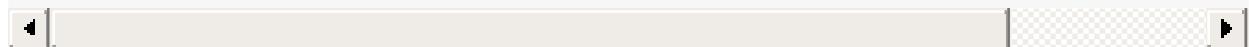
```

 count[task-'A']++;
 }

 // get the max_count

 // sort the vector, no need to care about relationship btw char->int
 sort(count.begin(), count.end());
 int max_count = count[25];
 int many_max = 0;
 int lens = tasks.size();
 // find how many same counts of max exist
 for (int i=25; i>=0; i--){
 if (count[i] == max_count) many_max++;
 else break;
 }
 return max(lens, (n+1) * (max_count - 1) + many_max); // dealing with edge cases
}
};


```



Follow-up:

Without changing the order:

```

class Solution {
public:
 int leastInterval(vector<char>& tasks, int n) {
 // count the char
 vector<int> count(26,0);
 for (char task: tasks){
 count[task-'A']++;
 }

 int max_count = 0;
 int many_max = 0;
 int lens = tasks.size();
 // get the max_count
 for (int i=0; i<26; i++){
 max_count = max(max_count, count[i]);
 }
 // get how many max_count are there in the count
 for (int i=0; i<26; i++){
 if (count[i] == max_count) many_max++;
 }
 return max(lens, (n+1) * (max_count - 1) + many_max); // max() is used to handle edge cases
 }
};


```



# 136. Single Number | 8/26

---

Given a non-empty array of integers, every element appears twice except for one. Find that single one.

Note:

Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

## Example 1:

**Input:** [2,2,1]

**Output:** 1

## Example 2:

**Input:** [4,1,2,1,2]

**Output:** 4

## 思路

XOR solution:

{2,1,4,5,2,4,1}

=> ((2<sup>^</sup>2)<sup>^</sup>(1<sup>^</sup>1)<sup>^</sup>(4<sup>^</sup>4)<sup>^</sup>(5)) => (0<sup>^</sup>0<sup>^</sup>0<sup>^</sup>5) => 5.

Note:  $2 \wedge 1 \wedge 2 = (2 \wedge 2) \wedge 1 = 0 \wedge 1$ .

## Code

```
class Solution {
public:
 int singleNumber(vector<int>& nums) {
 if (nums.empty()) return -1;
 unordered_map<int, int> counts; // integer<-> appearance time
 for (int i: nums){
 counts[i]++;
 }
 }
}
```

```
 for (auto c : counts){
 if (c.second == 1) return c.first;
 }
 return -1;
 }
};
```

XOR:

```
class Solution {
public:
 int singleNumber(vector<int>& nums) {
 int res = 0;
 for (int i: nums){
 res = res^i;
 }
 return res;
 }
};
```

## 282. Expression Add Operators | 8/26

---

Given a string that contains only digits 0-9 and a target value, return all possibilities to add binary operators (not unary) +, -, or \* between the digits so they evaluate to the target value.

### Example 1:

**Input:** num = "123", target = 6

**Output:** ["1+2+3", "1\*2\*3"]

### Example 2:

**Input:** num = "232", target = 8

**Output:** ["2\*3+2", "2+3\*2"]

### Example 3:

**Input:** num = "105", target = 5

**Output:** ["1\*0+5", "10-5"]

### Example 4:

**Input:** num = "00", target = 0

**Output:** ["0+0", "0-0", "0\*0"]

### Example 5:

**Input:** num = "3456237490", target = 9191

**Output:** []

## 思路

Note: 需要注意的是，如果輸入為"000",0的話，容易出現以下的錯誤：

Wrong : ["0+0+0", "0+0-0", "0+00", "0-0+0", "0-0-0", "0-00", "00+0", "00-0", "000", "0+00", "0-00", "000", "00+0", "00-0", "000", "000"]

Should eliminate 00 + 00

Correct : ["000","00+0","00-0","0+00","0+0+0","0+0-0","0-00","0-0+0","0-0-0"]

## Code

```
class Solution {
public:
 vector<string> addOperators(string num, int target) {
 vector<string> res;
 recursive(num, target, 0, 0, "", res);
 return res;
 }

 void recursive(string num, int target, long laststep, long curnum, string out,
 if (num.size() == 0 && curnum == target){
 res.push_back(out);
 return;
 }
 // use substring to walk through each number combination from input
 for (int i=1; i<= num.size(); i++){
 string cur = num.substr(0, i);
 if (cur.size() > 1 && cur[0] == '0') return; // eliminate edge case
 string next = num.substr(i); // update the new num
 if (out.size() > 0){
 recursive(next, target, +stoll(cur), curnum+stoll(cur), out + "+" +
 recursive(next, target, -stoll(cur), curnum-stoll(cur), out + "-" +
 recursive(next, target, laststep * stoll(cur), (curnum-laststep) +
 // Multiplication: 1+2*3 => curnum = (1+2), laststep = +2, the corr
 }
 else{
 recursive(next, target, stoll(cur), stoll(cur), cur, res);
 }
 }
 };
};
```

## 898. LeftMost One (Lintcode) | 8/26

- **@param arr:** The 2-dimension array
- **@return:** Return the column the leftmost one is located

## 思路

Haven't write it yet.

Leetcode don't have this question.

Lintcode need to pay.

## Code

```
public int getColumn(int[][] arr) {
 int leftMost = arr[0].length - 1;
 for (int r = 0; r < arr.length; r++) {
 // 每行不用从最右找，而是从之前行的left most找
 leftMost = binarySearch(arr[r], leftMost);
 if (leftMost <= 0) {
 return leftMost;
 }
 }
 return leftMost;
}

// 返回当前行的left most
public int binarySearch(int[] arr, int right) {
 int origRight = right;
 int left = 0;
 while (left + 1 < right) {
 int mid = (right - left) / 2 + left;
 if (arr[mid] == 1) {
 right = mid;
 } else {
 left = mid;
 }
 }
 if (arr[left] == 1) {
 return left;
 }
 if (arr[right] == 1) {
 return right;
 }
 return origRight;
}
```

## 301. Remove Invalid Parentheses | Re-do it later

In interview: only need to one valid parentheses

# 思路

## Code

```
```  
---  
## 560. Subarray Sum Equals K | 8/27  
Given an array of integers and an integer k, you need to find the total number of c
```

Example 1:

Input:nums = [1,1,1], k = 2

Output: 2

Note:

The length of the array is in range [1, 20,000].

The range of numbers in the array is [-1000, 1000] and the range of the integer k is [1, 1000].

思路

反省：要把題目看完整，整數的範圍是什麼

舉 testcase 的時候也要包含完整

1. 一開始想用queue去解這題，遇到over target的value就pop，但這題用queue完全行不通

2. 也沒辦法用two pointer來解

原因都是：輸入有包含負整數

當你把數pop掉時，未來有可能出現負數使得"你可以不用pop"

如 [1,2,3,4,-3,4,-3] k=7

當你的queue為[4 3 2 1] 你再來就會把1 pop掉，但其實[-3 4 3 2 1] 是個解。

神解法(紀錄曾經走過的路)：

```
  
### Code
```

c

```
class Solution {  
public:  
int subarraySum(vector& nums, int k) {  
int res = 0;  
int sum = 0;
```

```
unordered_map path;
path[0] = 1; // not eleminate any number, is also a solution, like [0,3,4] k=7
```

```
for (int i=0; i<nums.size(); i++){
    sum += nums[i];
    res += path[sum - k]; // is there any path I can eleminate, to reach target
    path[sum]++;
}
return res;
};

---
```

```
## 54. Spiral Matrix | 8/28
Given a matrix of m x n elements (m rows, n columns), return all elements of the ma

```

思路
Trick here: HOW TO INITIALIZE 2-D matrix

```
=> vector<vector<int>> path(row, vector<int> (column, 0));
```

本來想要用一個跟matrix一樣大小的二維vector去紀錄曾經走過的路線，

後來發現不用這麼麻煩，因為都是很規律地走。

不過也學習到如何初始化二維vector。

回到這題的思路：

這題的關鍵是，知道何時要轉彎，及更重要的，不再走過重複的點

右下左上的行走，分別是一個迴圈

那剩下的問題就剩下，標記走過的路線及何時停止

當往右走完時，再來要往下走，此時就是標記走過的時機，

方法是將上邊界下移一格，這樣再往上走時就不會再走到剛剛走過的這一條。

同理其他邊界。

如果不該超過的邊界，超過了，就是要break的時候了。

Code

```
C
/*
edge case:
```

```
[1],
[2], => [1,2,3]
[3]

[1,2,3] => [1,2,3]
[] => []
[2] => [2]
right -> down -> left -> up
```

```
*/
class Solution {
public:
vector spiralOrder(vector<>& matrix) {
if (matrix.empty() || matrix[0].empty()) return { };
vector res;
int row = matrix.size();
int column = matrix[0].size();
```

```
    int bR = column-1, bD = row-1, bL = 0, bU = 0; // set the boundary of each side
    while (true){
        // going right, when we bump into bR, we move the up boundary down.
        for (int i=bL; i <= bR; i++)      res.push_back(matrix[bU][i]);
        if (++bU > bD)  break; // move down the boundary up so that we don't bump i

        // going down, when we bump into bD, we move the right boundary left.
        for (int j=bU; j <= bD; j++)      res.push_back(matrix[j][bR]);
        if (--bR < bL)  break;

        // going left, when we bump into bL, we move the down boundary up.
        for (int i=bR; i >= bL; i--)      res.push_back(matrix[bD][i]);
        if (--bD < bU)  break;

        // going up, when we bump into bU, we move the left boundary right.
        for (int j=bD; j >= bU; j--)      res.push_back(matrix[j][bL]);
        if (++bL > bR)  break;
    }
    return res;
}

};
```

```
---
## 885. Spiral Matrix III | 8/28
On a 2 dimensional grid with R rows and C columns, we start at (r0, c0) facing east

Here, the north-west corner of the grid is at the first row and column, and the sou

Now, we walk in a clockwise spiral shape to visit every position in this grid.

Whenever we would move outside the boundary of the grid, we continue our walk outsi

Eventually, we reach all R * C spaces of the grid.

Return a list of coordinates representing the positions of the grid in the order th


### 思路
1. Right -> Down -> Left -> Up

2. We don't walk for the same steps; our steps inc for each two turn

3. Right * step -> Down * step .....; step += 1 for each two turn.

i. x = [0 ,1 ,0 ,-1]
y = [1 ,0 ,-1 , 0]

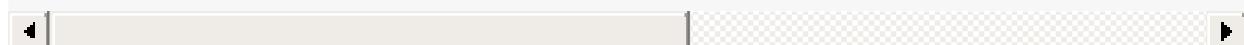
ii. module % 4 => each two turn, cur = (cur+1)%4

ii. when cur is 0 or 2 => we add our step.

iii. for-loop, to express total walk distance

Keep doing this, until the res is full which is R*C big.

### Code
```



```
c
class Solution {
public:
vector> spiralMatrixIII(int R, int C, int r0, int c0) {
vector> res;
res.push_back({r0, c0}); // the initial step; first time goin to the for-loop need to turn down, so
push it first
int cur=0, step=0;
int dx[4] = {0, 1, 0, -1};
int dy[4] = {1, 0, -1, 0};
```

```

        while (res.size() < R*C){
            if (cur == 0 || cur == 2)    step++;
            for (int i=0; i<step; i++){
                r0 += dx[cur]; c0 += dy[cur];
                if (r0 >= 0 && r0 < R && c0 >= 0 && c0 < C)
                    res.push_back({r0, c0});
            }
            cur = (cur+1) % 4;
        }
        return res;
    }
};


```

56. Merge Intervals | 8/28

Given a collection of intervals, merge all overlapping intervals.

 ### 思路

Trick here:

若要排序雙層vector，vector<vector<int>>，default 是按照內層的vector的第一 element

[[2,5],[1,3]] => [[1,3],[2,5]]

區間合併 基礎題

把第一個區間存入結果中，然後從第二個開始遍歷區間集，如果結果中最後一個區間和遍歷的當前區間無重疊，直
 ### Code



```

C
class Solution {
public:
vector> merge(vector>& intervals) {
vector> out;
if (intervals.empty()) return out;
sort(intervals.begin(), intervals.end());
out.push_back(intervals[0]);

```

```

for (int i=1; i<intervals.size(); i++){
    if (out.back()[1] < intervals[i][0]){
        out.push_back(intervals[i]);
    }
    else {

```

```

        out.back()[1] = max(out.back()[1], intervals[i][1]);
    }
}
return out;
}

};


```

*## ***[Start to Leetcode on high frequency question on tag:Amazon interview]****

1. Two Sum / 8/29

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, and you may not use the same element twice.

Example:

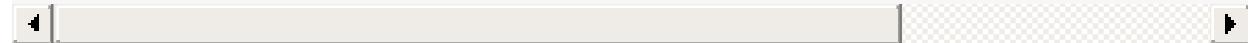
Given nums = [2, 7, 11, 15], target = 9,

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

return [0, 1].

思路

Code



```

C
class Solution {
public:
vector<int> twoSum(vector<int>& nums, int target) {
vector<int> res;
unordered_map<int, int> diff_map;
for (int i=0; i<nums.size(); i++){
diff_map[nums[i]] = i;
}


```

```

// can't use same value
for (int i=0; i<nums.size(); i++){
    int diff = target - nums[i]; // i-1 is the fixed number
    if (diff_map[diff] != 0 && diff_map[diff] != i){
        res.push_back(i);
        res.push_back(diff_map[diff]);
        break;
    }
}


```

```
        }
    }
    return res;
}

};


```

```
---
## ***[Start to Leetcode with Python]***
---

## 146. LRU Cache | 8/29
Design and implement a data structure for Least Recently Used (LRU) cache. It should

get(key) - Get the value (will always be positive) of the key if the key exists in
put(key, value) - Set or insert the value if the key is not already present. When t

The cache is initialized with a positive capacity.
```

Follow up:

Could you do both operations in O(1) time complexity?

Example:

```

```

思路

- 用dummy和tail去指向linklist的頭尾
- 用雙向指針去刪除節點
- 要拿entryFinder的值用 .get(key)
- 刪除entryFinder的值用 del entryFinder[key]

dummy -> entry -> entry -> entry -> entry(tail)

entryFinder = { key, {key, value} }

Code



py

...

dummy -> entry -> entry -> entry -> entry(tail)

entryFinder = { key, {key, value} }

...

class Node:

def init(self, key, value):

self.key = key

self.value = value

```
self.prev = None  
self.next = None
```

```
class LRUCache(object):
```

```
def __init__(self, capacity):  
    """  
    :type capacity: int  
    """  
    self.capacity = capacity  
    self.size = 0 # Current size of the cache (linklist)  
    self.dummy = Node(-1,-1)  
    self.tail = self.dummy # Point to the end of list  
    self.entryFinder = {} # { key, {key, value} }  
  
def get(self, key):  
    """  
    :type key: int  
    :rtype: int  
    """  
    entry = self.entryFinder.get(key)  
    if entry is not None:  
        self.renew(entry)  
        return entry.value  
    else:  
        return -1  
  
def put(self, key, value):  
    """  
    :type key: int  
    :type value: int  
    :rtype: None  
    """  
    entry = self.entryFinder.get(key)  
    if entry is None:  
        entry = Node(key, value)  
        self.entryFinder[key] = entry  
  
        # link new node to the tail of list  
        self.tail.next = entry  
        entry.prev = self.tail  
        self.tail = entry  
  
        # Check if over the capacity  
        if self.size < self.capacity:  
            self.size += 1  
        else:  
            # Remove the Node at the most front of the list  
            head = self.dummy.next  
            if head is not None:  
                self.dummy.next = head.next  
                head.next.prev = self.dummy  
            # Remove it from dictionary
```

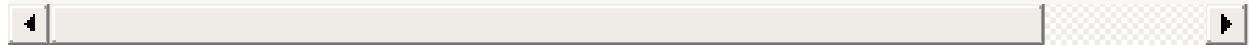
```

        del self.entryFinder[head.key]
    else:
        # Put the renew data, whose key exist but the value update, to the tail of
        entry.value = value
        self.renew(entry)

    # Move the used data to the end of the list
def renew(self, entry):
    if self.tail != entry:
        # delete(jump over) the entry and linked
        prevNode = entry.prev
        nextNode = entry.next
        prevNode.next = nextNode
        nextNode.prev = prevNode

        # link the entry to the tail of the list
        entry.next = None
        self.tail.next = entry
        entry.prev = self.tail
        self.tail = entry

```



4. Median of Two Sorted Arrays / 8/30

There are two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively.

Find the median of the two sorted arrays. The overall `run time` complexity should be

You may assume `nums1` and `nums2` cannot be both empty.

Example 1:

```
nums1 = [1, 3]
nums2 = [2]
```

The median is `2.0`

Example 2:

```
nums1 = [1, 2]
nums2 = [3, 4]
```

The median is `(2 + 3)/2 = 2.5`

技巧


```
>>> word[0:2]  # characters from position 0 (included) to 2 (excluded)
'Py'
>>> word[2:5]  # characters from position 2 (included) to 5 (excluded)
'tho'
```

```
Note how the start is always included, and the end always excluded. This makes sure
>>> word[:2]    # character from the beginning to position 2 (excluded)
'Py'
>>> word[4:]    # characters from position 4 (included) to the end
'on'
>>> word[-2:]   # characters from the second-last (included) to the end
'on'

+---+---+---+---+---+
| P | y | t | h | o | n |
+---+---+---+---+---+
  0   1   2   3   4   5   6
 -6  -5  -4  -3  -2  -1
```

思路

這道題要求兩個已經排好序的數列的中位數。中位數的定義：如果數列有偶數個數，那麼中位數為中間兩個數的平均值。

首先我們來看如何找到兩個數列的第k小個數，即程序中getKth(A, B, k)函數的實現。用一個例子來說明這個問題。

```

```

Code



py

```
class Solution(object):
    def findKth(self, A, B, k):
        lenA = len(A); lenB = len(B)
        # fixed the B is the larger one
        if lenA > lenB: return self.findKth(B, A, k)
```

```
        if lenA == 0:      return B[k-1]
        if k == 1:         return min(A[0], B[0])
        pa = min(k/2, lenA); pb = k-pa
        if A[pa-1] <= B[pb-1]:
            return self.findKth(A[pa:], B, pb)
        else:
            return self.findKth(A, B[pb:], pa)
```

```
def findMedianSortedArrays(self, nums1, nums2):
    .....
    :type nums1: List[int]
    :type nums2: List[int]
    :rtype: float
    .....
    lenA = len(nums1); lenB = len(nums2)
    if (lenA+lenB) % 2 == 1:
        return self.findKth(nums1, nums2, (lenA+lenB)/2 +1)
```

```
        else:
            return (self.findKth(nums1, nums2, (lenA+lenB)/2) + self.findKth(nums1, num
```

--
5. Longest Palindromic Substring / 8/31
Given a string s, find the longest palindromic substring in s. You may assume that

思路

中心思想就是用迴圈去找到回文的中心點

並把回文長度分成奇數偶數兩個case去看

Time complexity O(n^n)

最佳解：

馬拉車算法

<https://www.cnblogs.com/grandyang/p/4475985.html>

Code

Naive Solution:

```
py
class Solution(object):
    def longestPalindrome(self, s):
        res = ""
        """
        :type s: str
        :rtype: str
        """

        for i in range(len(s)):
            # check odd case
            temp = self.checkLR(s, i, i)
            if len(temp) > len(res):
                res = temp
```

```
# check even case
temp = self.checkLR(s, i, i+1)
if len(temp) > len(res):
    res = temp
```

```
    return res

def checkLR(self, s, l, r):
    while l >= 0 and r < len(s) and s[l] == s[r]:
        l -= 1; r += 1
    return s[l+1 : r]
```

```
---
## 200. Number of Islands / 8/31
Given a 2d grid map of '1's (land) and '0's (water), count the number of islands. A

```

技巧

1. 初始化二维矩阵: `[[0]*n for i in range m] -> [[0,0,...n], [0,0,...n]...m]`

<https://blog.csdn.net/u012814856/article/details/78740043>

2. 封装: `dirz = zip([1,0,-1,0], [0,1,0,-1]) -> dirz = [(1,0),(0,1),(-1,0),(0,-1)]`
(在pair操作上可以用)

<http://www.runoob.com/python/python-func-zip.html>

3. 可以直接用 `m > k >= 0`

4. Use List as Stack: stack用pop()

5. Use List as Queue: queue用pop(0) -> slower than using deque

5. Use deque as Queue: queue = deque([List])
- `from collections import deque`
- `queue.append(), queue.popleft()`

思路

Naive solution

v.s.

clean and clear solution

Code



py

```
class Solution(object):
def numIslands(self, grid):
    """
```

```

:type grid: List[List[str]]
:rtype: int
"""

ans = 0
if not len(grid): return ans

    m, n = len(grid), len(grid[0])
    visited = [[False]*n for x in range(m)] # [[False, False, ... n], [False, False
    for x in range(m):
        for y in range(n):
            if grid[x][y] == '1' and not visited[x][y]:
                ans += 1
                self.bfs(grid, visited, x, y, m, n)
    return ans

def bfs(self, grid, visited, x, y, m, n):
    dirz = zip([1,0,-1,0], [0,1,0,-1]) # the direction it walk
    queue = [(x,y)]
    visited[x][y] = True

    while queue:
        front = queue.pop(0)
        for p in dirz: # why not "for p in range(dirz)"
            # px = p[0]+front[0], py = p[1]+front[1]      Wrong!!
            np = (front[0]+p[0], front[1]+p[1])
            if self.isValid(np, m, n) and not visited[np[0]][np[1]] and grid[np[0]][
                np[1]] == '1':
                queue.append(np)
                visited[np[0]][np[1]] = True

    def isValid(self, np, m, n):
        return m > np[0] >= 0 and n > np[1] >= 0

```

Beautiful code:

```

py
class Solution(object):
    def numIslands(self, grid):
        if not grid:
            return 0

```

```

        count = 0
        for i in range(len(grid)):
            for j in range(len(grid[0])):
                if grid[i][j] == '1':

```

```

        self.dfs(grid, i, j)
        count += 1
    return count

def dfs(self, grid, i, j):
    if i<0 or j<0 or i>=len(grid) or j>=len(grid[0]) or grid[i][j] != '1':
        return
    grid[i][j] = '#'
    self.dfs(grid, i+1, j)
    self.dfs(grid, i-1, j)
    self.dfs(grid, i, j+1)
    self.dfs(grid, i, j-1)

```

42. Trapping Rain Water / 9/1

Given n non-negative integers representing an elevation map where the width of each

技巧

從後面loop回來 `for(i in range(n-1, -1, -1))` -> 從 `arr[n-1] ~ arr[0]`

思路

我們觀察其中的一個位置，單獨考慮這個位置的容量是多少？

這個位置左邊最高的那個邊(含自己)，和右邊最高的那個邊(含自己)，兩者取小的，

然後再減去本身的大小，那麼結果就是這個位置的容量。所以，

- (1) 從左向右進行掃瞄，獲取每個位置的左邊最高的邊。
- (2) 從右向左進行掃瞄，獲取每個位置的右邊最高的邊。
- (3) 再遍歷一邊，計算出每個位置的容量，累加，即結果。

Code



py

```
class Solution(object):
```

```
def trap(self, height):
```

```
    """
```

```
:type height: List[int]
```

```
:rtype: int
```

```
"""
```

```
if not height: return 0
```

```
res, n = 0, len(height)
left_max = [0] * n # the left highest of the point
right_max = [0] * n # the right hightest of the point

left_max[0] = height[0] # initialize
for i in range (1,n):
    left_max[i] = max(left_max[i-1], height[i])

right_max[n-1] = height[n-1]
for i in range (n-2, -1, -1):
    right_max[i] = max(right_max[i+1], height[i])

for i in range (1,n-1):
    res += min(left_max[i], right_max[i]) - height[i]

return res
```

```
---
```

3. Longest Substring Without Repeating Characters / 9/1
Given a string, find the `length` of the longest substring without repeating characters

```

```

技巧

判斷字典裡存不存在: if s[i] in Dic: Dic[s[i]] += 1

思路

利用字典紀錄目前此char的最高位置

利用滑動窗口原理，並只maintain tail

當 i (head)持續前進，

如果遇到已經存在的，

把前面吃下來，並且移動tail到 重複字元在字典中的位置+1

有可能會讓字串變小，但我們持續記錄最長字串所以無妨

Code

py

```
class Solution(object):
    def lengthOfLongestSubstring(self, s):
        """
        :type s: str
        :rtype: int
        """

        charPos = {} # maintain the newest position of each char
        tail, ans = 0, 0
```

```
# tail is the end of current str, i is the head
for i in range(len(s)):
    if s[i] in charPos and tail <= charPos[s[i]]: # avoid move the tail to front
        tail = charPos[s[i]] + 1
    else:
        ans = max(ans, i - tail + 1)

    charPos[s[i]] = i # update the char
return ans
```

運用`enumerate`改寫版：

```
py
class Solution(object):
    def lengthOfLongestSubstring(self, s):
        """
        :type s: str
        :rtype: int
        """

        charPos = {} # maintain the newest position of each char
        tail, ans = 0, 0
```

```
# tail is the end of current str, i is the head
for i, c in enumerate(s):
    if c in charPos and tail <= charPos[c]: # avoid move the tail to front
        tail = charPos[c] + 1
    else:
        ans = max(ans, i - tail + 1)

    charPos[c] = i # update the char
return ans
```

```
--  
## 973. K Closest Points to Origin / 9/1  
We have a list of points on the plane. Find the K closest points to the origin (0,  
(Here, the distance between two points on a plane is the Euclidean distance.)  
You may return the answer in any order. The answer is guaranteed to be unique (exc  

```

技巧

終於自己用py寫出一題了！雖然TLE

0. enumerate: 在loop一個iterable的object，導入此函數可以讓你用index去存取

- for i, point in enumerate(points):
- print i, point => 0, (0,0) 1,(0,1) ...
- for i, point in enumerate(points, 1):
- print i, point => 1, (0,0) 2,(0,1) ...
- 目前想到可用於陣列向後位移k個單位
- 在第三題有個示範

1. heapq

2. 把點座標當作是dic的索引: dic[(list[0], list[1])]

3. 最大值: sys.maxint, 最小值: -(sys.maxint) -1

4. for entry in dict

思路

實作只取出前K小的數，雖然TLE不過還是可以學習

先取出最小，把最小刪掉，再搜索一次 => O(n!)

Code

TLE: (75 / 83 test cases passed.)



py

```
class Solution(object):  
    def kClosest(self, points, K):  
        """  
        :type points: List[List[int]]  
        :type K: int  
        :rtype: List[List[int]]  
        """
```

```

ans = []
dic = {}
for point in points:
    dic[(point[0], point[1])] = point[0]**2 + point[1]**2

ansout = (-1,-1)
for i in range(K):
    smallest = sys.maxint
    for entry in dic:
        if dic[entry] < smallest:
            smallest = dic[entry]
            ansout = entry
    ans.append(ansout)
    del dic[ansout]

return ans

```

Heapq 解法

```

py
from heapq import heappush, heappop
class Solution(object):
    def kClosest(self, points, K):
        """
        :type points: List[List[int]]
        :type K: int
        :rtype: List[List[int]]
        """
        heap = []
        for i, point in enumerate(points):
            val = point[0]**2 + point[1]**2
            heappush(heap, (-val, point))
            if i >= K:
                heappop(heap)

```

```
        return [e[1] for e in heap]
```

53. Maximum Subarray | 9/2

Given an integer array nums, find the contiguous subarray (containing at least one

Example:

Input: [-2,1,-3,4,-1,2,1,-5,4],

Output: 6

Explanation: [4,-1,2,1] has the largest sum = 6.

Follow up:

If you have figured out the $O(n)$ solution, try coding another solution using the di

思路

1, Directly approach:

Compare this element vs this element + pastMax combination (curSum)

2, Divide and Conquer:

Find the maximum subarray in nums[start ~ mid - 1].

Find the maximum subarray in nums[mid + 1 ~ end].

Find the maximum subarray which includes nums[mid];

Return the max of the above three.

Time complexity: We use $O(n)$ operation and leave two $T(n/2)$ problems.

$T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n\log n)$

Code



py

```
class Solution(object):
    def maxSubArray(self, nums):
```

"""

:type nums: List[int]

:rtype: int

"""

```
    curSum, res = -(sys.maxint)-1, -(sys.maxint)-1 # Or can set as nums[0]
```

```
    for num in nums: # And loop from nums[1] -> in nums[1:]:
```

```
        curSum = max(num+curSum, num)
```

```
        res = max(res, curSum)
```

```
    return res
```

Divide and Conquer Way:

```
py
class Solution(object):
def maxSubArray(self, nums):
"""
:type nums: List[int]
:rtype: int
"""

return self.DandC(0, len(nums)-1, nums)
```

```
def DandC(self, left, right, nums):

    if left >= right:    return nums[left]
    mid = left + (right-left)/2
    lmax = self.DandC(left, mid, nums) # find max in left
    rmax = self.DandC(mid+1, right, nums) # find max in right

    # find max includes nums[mid]
    mmax, t = nums[mid], nums[mid]
    for i in range(mid-1, left-1, -1):
        t += nums[i]
        mmax = max(mmax, t)

    t = mmax
    for i in range(mid+1, right+1):
        t += nums[i]
        mmax = max(mmax, t)

    return max(lmax, rmax, mmax)
```

20. Valid Parentheses / 9/2
Given a string containing just the characters '(', ')', '{', '}', '[', and ']', determine if the input string is valid.

An input string is valid if:

- Open brackets must be closed by the same type of brackets.
- Open brackets must be closed in the correct order.

Note that an empty string is also considered valid.

```

```

技巧

直接判斷dictionary裡面有沒有特定key或value:

```
- if char in dict.keys():
- if char in dict.values():
```

思路

雙Stack法:

先把原輸入推進inStack中，

再從inStack一個個pop出來，

如果pop出的是後標，先放進outStack中，

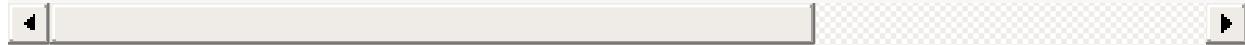
如果pop出的是前標，去outStack的頂端看是不是對應的前標，

如果不是，就是invalid

如果inStack run

Code

雙Stack法:



py

```
class Solution(object):
def isValid(self, s):
"""
:type s: str
:rtype: bool
"""

if not s: return True
dic = {'{': '}', '[': ']', '(': ')'} 
```

```
inStack = []
outStack = []
for c in s:
    inStack.append(c)

while inStack:
    out = inStack.pop()
    if out == '}' or out == ']' or out == ')':
        outStack.append(out)
```

```

        elif out == '[' or out == '(' or out == '{':
            if not outStack:    return False
            elif outStack.pop() != dic[out]: return False

        if not outStack:
            return True
        else:
            return False

```

大神版：

```

py
class Solution:
# @return a boolean
def isValid(self, s):
stack = []
dict = {"}": "[", "}": "{", ")": "("}
for char in s:
if char in dict.values():
stack.append(char)
elif char in dict.keys():
if stack == [] or dict[char] != stack.pop():
return False
else:
return False
return stack == []

```

937. Reorder Log Files / 9/2
You have an array of logs. Each log is a space delimited string of words.

For each log, the first word in each log is an alphanumeric identifier. Then, either
Each word after the identifier will consist only of lowercase letters, or;
Each word after the identifier will consist only of digits.
We will call these two varieties of logs letter-logs and digit-logs. It is guaranteed
Reorder the logs so that all of the letter-logs come before any digit-log. The letter-logs
Return the final order of the logs.

技巧

1. 客製化sort: list.sort(key=lambda x: {每個在list裡的元素x的排序規則})

2. `str.split()`: 會依照空格把字串切開，並回傳`list`格式

3. 思考：`sort`的順序(如程式碼末端)

Code



py

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

if not logs: return []
letter = []
digit = []
```

```
for log in logs:
    if log.split()[1].isdigit(): # the first str after the identifier
        digit.append(log)
    else:
        letter.append(log)

# firstly, sort by context, but
# if the context is tie, sort by identifier

# Note: in code, we have to sort by identifier first
# Otherwise the outcome would reversely as our think
letter.sort(key = lambda x: x.split()[0] )
letter.sort(key = lambda x: x.split()[1:])

return letter+digit
```

238. Product of Array Except Self / 9/2

Given an `array` `nums` of `n` integers where `n > 1`, return an `array` output such that ou

Example:

Input: [1,2,3,4]

Output: [24,12,8,6]

Note: Please solve it without division and in O(n).

Follow up:

Could you solve it with constant space complexity? (The output array does not count
路思

___思考如何求出特定的數 A[3]___

對於某一個數字，如果我們知道其前面所有數字的乘積，同時也知道後面所有的數乘積，那麼二者相乘就是我們要

___如果要求constant space，表示不能有除了回傳陣列外多餘的陣列___

由於最終的結果都是要乘到結果 res 中，所以可以不用單獨的數組來保存乘積，而是直接累積到結果 res 中，

Code



py

'''

0 1 2 3 4 5 6 n-1

```
A[3] = (A[0]*..A[2]) * (A[4]*...A[n-1])
      fwd           bwd
```

'''

```
class Solution(object):
    def productExceptSelf(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]
        """
        n = len(nums)
        fwd, bwd = [1]*n,[1]*n
        res = []
        for i in range(1, n):
            fwd[i] = fwd[i-1]*nums[i-1]
```

```
        for i in range(n-2, -1, -1):
            bwd[i] = bwd[i+1]*nums[i+1]

        for i in range(n):
            res.append(bwd[i]*fwd[i])
        return res
```

Constant space improve:

```
py
class Solution(object):
    def productExceptSelf(self, nums):
        """
:type nums: List[int]
:rtype: List[int]
"""

n = len(nums)
res = [1]*n
```

```
        for i in range(1, n):
            res[i] = res[i-1] * nums[i-1]

        afterMult = 1
        for i in range(n-2, -1, -1):
            afterMult *= nums[i+1]
            res[i] *= afterMult

        return res
```

7. Reverse Integer / 9/4
Given a 32-bit signed integer, reverse digits of an integer.

Example 1:

Input: 123

Output: 321

Example 2:

Input: -123

Output: -321

Example 3:

Input: 120

Output: 21

Note:

Assume we are dealing with an environment which could only store integers within the 32-bit signed integer range: $[-2^{31}, 2^{31} - 1]$.

Code

My code beat 96%

py

```
from collections import deque
```

```
class Solution(object):
```

```
    def reverse(self, x):
```

```
        """
```

```
:type x: int
```

```
:rtype: int
```

```
        """
```

```
        res = 0
```

```
        queue = deque()
```

```
        symbol = 1
```

```
        if (x < 0):
```

```
            symbol = -1
```

```
        x = -x
```

```
        while x != 0:
            queue.append(x%10)
            x /= 10
        while queue:
            res += queue.popleft() * pow(10, len(queue))

        if (res > pow(2,31) - 1 or res < pow(2,31) * -1):
            return 0
        else:
            return res*symbol
```

Super smart:

py

```
class Solution:
```

```
    # @return an integer
```

```
    def reverse(self, x):
```

```
        result = 0
```

```

if x < 0:
    symbol = -1
    x = -x
else:
    symbol = 1

while x:
    result = result * 10 + x % 10
    x /= 10

return 0 if result > pow(2, 31) else result * symbol

```

121. Best Time to Buy and Sell Stock / 9/4

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 (i.e., buy one and sell one stock), what would be the maximum profit?

Note that you cannot sell a stock before you buy one.

Example 1:

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

Output: 5

Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 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

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

思路

Code



```

py
class Solution(object):
def maxProfit(self, prices):
"""
:type prices: List[int]
:rtype: int

```

```
"""
maxprofit, minPrice = 0, float('inf')
for price in prices:
    minPrice = min(minPrice, price)
    profit = price - minPrice
    maxprofit = max(maxprofit, profit)
return maxprofit
```

Almost the same

```
py
class Solution(object):
    def maxProfit(self, prices):
        """
:type prices: List[int]
:rtype: int
"""

        maxprofit, minPrice = 0, float('inf')
        for price in prices:
            #minPrice = min(minPrice, price)
            if price < minPrice:
                minPrice = price
            else:
                profit = price - minPrice
            maxprofit = max(maxprofit, profit)
        return maxprofit
```

--

289. Game of Life / 9/6

According to the Wikipedia's article: "The Game of Life, also known simply as Life,

Given a board with m by n cells, each cell has an initial state live (1) or dead (0)

1. Any live cell with fewer than two live neighbors dies, as if caused by under-population.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by over-population.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

Write a function to compute the next state (after one update) of the board given its current state.

```
![] (assets/markdown-img-paste-20190906163727321.png)
```

Follow up:

Could you solve it in-place? Remember that the board needs to be updated at the same time.

In this question, we represent the board using a 2D array. In principle, the board

思路

....

Original thinking:

Use 2D bool array to record change or not
can use XOR

After reading the discussion:

We need two other states to tag "Change needed status"

Which is live-to-die and die-to live

But we should also consider the "states not change"

so how can we do that? We use "Mod" to do both in a time

0: mean died, 1 mean alive

So we want the changed board(not Mod yet) to become the result of Mod we want

2 % 2 = 0 -> so 2 mean live-to-died

3 % 2 = 1 -> so 3 mean died-to-live

....

Code



py

class Solution(object):

def gameOfLife(self, board):

"""

:type board: List[List[int]]

:rtype: None Do not return anything, modify board in-place instead.

"""

M, N = len(board), len(board[0])

dx = [1, 1, 0, -1, -1, 0, 1, -1]

dy = [0, 1, 1, 0, -1, -1, -1, 1]

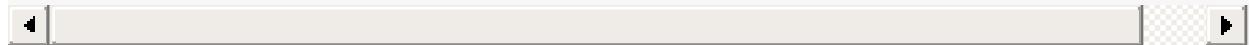
```
for m in range(M):
    for n in range(N):
        count = 0
        for i in range(8):
            x = m + dx[i]
            y = n + dy[i]
            # if is alive in cur state(no matter will die or not)
            if self.isValid(x,y,M,N) and (board[x][y] == 1 or board[x][y] == 2)
                count += 1
```

```

# if is alive, but need to die
if (board[m][n] == 1 or board[m][n] == 2) and (count < 2 or count > 3):
    board[m][n] = 2
# if is died, but need to revive
elif (board[m][n] == 0 or board[m][n] == 3) and (count == 3):
    board[m][n] = 3
# mod back to output
for m in range(M):
    for n in range(N):
        board[m][n] %= 2

def isValid(self, x, y, M, N):
    return 0 <= x < M and 0 <= y < N

```



49. Group Anagrams / 9/7

Given an array of strings, group anagrams together.

Note:

All inputs will be in lowercase.
The order of your output does not matter.

技巧

```

sort a string in python: sorted_str = "".join(sorted(str))

sorted("apple") => ['a', 'e', 'l', 'p', 'p']

"-".join(['a', 'b', 'c']) => "a-b-c"

```

思路

.....

We want to sort string.

but python string is unsortable (list is sortable)

except using built-in func: sorted(),

however, it will return list of the char after sort,

sorted("apple") => ['a', 'e', 'l', 'p', 'p']

So, we'd use join(), to help us collect them back to string

"-".join(['a', 'b', 'c']) => "a-b-c"

.....

Code

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

        _map = {}
        ret = []
        for _str in strs:
            key = "".join(sorted(_str))

            if key in _map.keys():
                _map[key].append(_str)
            else:
                _map[key] = [_str] # if want to use 'append' should initial first

        #return [entry for entry in _map.values()] # Can use the code below
        return _map.values()
```

Use defaultdict to assign initial status

```
py
class Solution(object):
    def groupAnagrams(self, strs):
        from collections import defaultdict
        """

        :type strs: List[str]
        :rtype: List[List[str]]
        """

        ret = defaultdict(list)

        for _str in strs:
            key = "".join(sorted(_str))
            ret[key].append(_str)

        return ret.values()
```

```
---
```

```
## 336. Palindrome Pairs | 9/9
```

```
Given a list of unique words, find all pairs of distinct indices (i, j) in the give
```

```

```

```
### 技巧
```

1. 反轉字串: `rev_str = str[::-1]`
2. 切割字串:
 - `n = len("apple")`
 - `str[:2] = "ap", str[2:] = "ple"`
 - `str[:n] = "apple", str[n:] = ""` → 看起來會overflow但不會. cuz there is a empty
3. 少用指定搜尋`dic.keys()`, 會TLE
 - 直接用: `if reverse != word and __reverse in dic__:`

```
### 思路
```

```

```

```
The basic idea is to check each word for prefixes (and suffixes) that are themselves
```

```
words = ["bot", "t", "to"]
```

```
Starting with the string "bot". We start checking all prefixes. If "", "b", "bo", "
```

```
You can do the same thing by checking all suffixes to see if they are palindromes.
```

```
The process is then repeated for every word in the list. Note that when considering
```

```
### Code
```



```
py
```

```
class Solution(object):  
    def palindromePairs(self, words):
```

```
        """
```

```
:type words: List[str]
```

```
:rtype: List[List[int]]
```

```
        """
```

```
        dic = { }  
        ret = []  
        for i, word in enumerate(words): #簡寫成 dic = {word: i for i, word in enumerate  
            dic[word] = i  
  
        for i, word in enumerate(words):  
            n = len(word)  
            for j in range(n+1):  
                pre = word[:j] # the current prefix  
                suf = word[j:] # the current suffix
```

```

# fix the prefix, find reverse(sufix)
if self.is_palind(pre):
    reverse = suf[::-1]
    if reverse != word and reverse in dic:
        ret.append([dic[reverse], i])

# fix the sufix, find reverse(prefix)
if self.is_palind(suf) and j != n: # duplicate find the whole word
    reverse = pre[::-1]
    if reverse != word and reverse in dic:
        ret.append([i, dic[reverse]])

return ret

def is_palind(self, string):
    return string == string[::-1] # reverse the string

```



11. Container With Most Water / 9/10

Given n non-negative integers a_1, a_2, \dots, a_n , where each represents a point at co

Note: You may not slant the container and n is at least 2.

 ### 思路

Since we know $\text{area} = \text{length} * \min(\text{height}_a, \text{height}_b)$, to maximize the area we wan

Width: We set two pointers, which are initialized as 0 and $\text{len}(\text{height}) - 1$ to get t

Height: We move the left pointer and right pointer respectively to search for the n

有兩種最大值要找的時候，

先固定一種(寬度最大)，

然後一步一步向內縮，

同時持續搜尋最大值

Code



py

```

class Solution(object):
    def maxArea(self, height):
        """
:type height: List[int]
:rtype: int
"""
        j = len(height) - 1

```

```
i = 0
res = 0
while i < j:
    res = max(res, min(height[i],height[j]) * (j-i))
    if height[i] < height[j]:
        i+=1
    else:
        j-=1
```

```
return res
```

```
## 10. Regular Expression Matching (HARD) / 8/29
Given an input string (s) and a pattern (p), implement regular expression matching.
```

'.' Matches any single character.

'*' Matches zero or more of the preceding element.

The matching should cover the entire input string (not partial).

Note:

s could be empty and contains only lowercase letters a-z.

p could be empty and contains only lowercase letters a-z, and characters like . or *

```


```

思路

DP

Code



py

```
## ***[Start to LinkedIn High Freq]***
```

```
## 380. Insert Delete GetRandom O(1) / 9/10
Design a data structure that supports all following operations in average O(1) time
```

insert(val): Inserts an item val to the set if not already present.

```
remove(val): Removes an item val from the set if present.
```

```
getRandom: Returns a random element from current set of elements. Each element must  

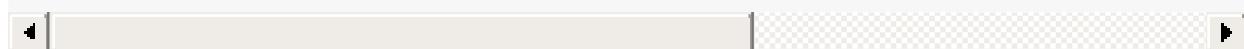
```

技巧

1. 字典的set(), get() 都是O(1)
2. 如何刪除一個list只用O(1)複雜度?
 - 多用一個字典紀錄List element的位置
 - 刪除時先找出target的位置
 - 將target位置的值換成List最後一位的值
 - 更新字典中最後一位值的位置, (成原位置的值)
 - Pop list + del 字典的entry

思路

Code



py

"""

Your RandomizedSet object will be instantiated and called as such:

```
obj = RandomizedSet()  
param_1 = obj.insert(val)  
param_2 = obj.remove(val)  
param_3 = obj.getRandom()
```

"""

```
class RandomizedSet(object):
```

```
def __init__(self):  
    """  
    Initialize your data structure here.  
    """  
    self.num = [] # the list of the number  
    self.pos = {} # the position of each number in the list  
  
def insert(self, val):  
    """  
    Inserts a value to the set. Returns true if the set did not already contain the  
    :type val: int  
    :rtype: bool  
    """  
    if val not in self.pos:  
        self.num.append(val)  
        self.pos[val] = len(self.num) - 1  
        return True  
    return False
```

```

def remove(self, val):
    """
    Removes a value from the set. Returns true if the set contained the specified element.
    :type val: int
    :rtype: bool
    """

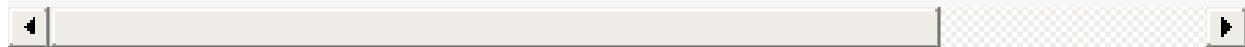
    if val in self.pos:
        idx = self.pos[val] # get the target index
        last = self.num[-1] # get the last number in the list
        self.num[idx] = last # replace the target with last number
        self.pos[last] = idx # update the position with it's new home

        self.num.pop() # pop the last element(which has already moved to new house)
        del self.pos[val]
        return True
    return False

def getRandom(self):
    """
    Get a random element from the set.
    :rtype: int
    """

    import random
    return random.choice(self.num)

```



```

---
## 324. Wiggle Sort II | 9/18
Given an unsorted array nums, reorder it such that nums[0] < nums[1] > nums[2] < nums[3] < ...


### 技巧
- 切片(Slice): arr[start:inc/dec], **由inc/dec决定方向**
  - e.g. nums = [1,2,3,4,5]
  - Step是正數
    - print(nums) # 12345
    - print(nums[::-2]) # 135
    - print(nums[1::2]) # 24
  - Step是負數
    - counts from the end of the array instead of the beginning. So:
      - a[-2:] # last two items in the array
      - a[:-2] # everything except the last two items
      - Similarly, step may be a negative number:
        - a[::-1] # all items in the array, reversed
        - a[1::-1] # the first two items, reversed
        - a[:-3:-1] # the last two items, reversed
        - a[-3::-1] # everything except the last two items, reversed
        - print(nums[2::-1]) # 321
        - print(nums[:2:-1]) # 54
- 關於不指定的部分

```

- decrease的狀況
 - 在end不指定，就是指最前端
 - 在start不指定，就是指最末端
- increase的狀況
 - 在end不指定，就是指最末端
 - 在start不指定，就是指最前端
- //////
 - 這行：`nums[::2]`, `nums[1::2] = nums[mid::-1]`, `nums[:mid:-1]`
 - 不等於這兩行，why?
 - `nums[::2] = nums[mid::-1]`
 - `nums[1::2] = nums[:mid:-1]`
 - 因為當你執行完第一行時，`nums`裡的值已經被改變了！

路徑
我們可以先給數組排序，然後在做調整。調整的方法是找到數組的中間的數，相當於把有序數組從中間分成兩部分

follow-up: $T = O(n) \rightarrow$ 不能sort，解法有點複雜...

Code



```
py
```

```
"""
```

Discuss how to find our mid point:

odd num: 5: 1 2 3 4 5

odd digit:2 mid at idx= 2 -> 5,4

even digit:3 -> 3,2,1

even num: 4: 1 2 3 4

odd digit: 2 mid at idx=1 -> 4,3

even digit:2 -> 2,1

```
"""
```

```
class Solution(object):
```

```
    def wiggleSort(self, nums):
```

```
        """
```

```
:type nums: List[int]
```

```
:rtype: None Do not return anything, modify nums in-place instead.
```

```
"""

mid = (len(nums)+1) / 2 -1 # 4/2=2, 5/2=2 -> floor(5/2)
nums.sort()
nums[::2], nums[1::2] = nums[mid::-1], nums[:mid:-1]
```

```
---
```

```

## ***[Start to Twitter High Freq]***
---
## Twitter. New Office Design | 9/19

給兩個int陣列分別代表位置 及 高度

pos[1, 10], hight[1, 5]

我們的任務是要在所有填空中回傳最高的填空高度，

填空： 1跟10中間有2~9的空可以填

限制：每個位置的高度，不能超過左右兩邊1

pos'[1,2,3,4,5,6,7,8,9,10]

hig'[1,2,3,4,5,6,7,7,6,5]

回傳 7

[](https://leetcode.com/discuss/interview-question/373110/Twitter-or-0A-2019-or-New
### 思路
- Naive:
  - 用一個HashMap對應pos跟height
  - 計算出每個位置，回傳最高的

  -  $O(n)$ 解法，（假設pos有排序過）
    - The idea is to calculate maxHeight, given two heights and empty slots between them.
      - [1] If both heights are same then meet in middle.
      - [2] If they are not same then
        - [2.A] check if we can make it same by advancing smaller height further and
        - [2.B] we cannot make it same , then its simply small height + empty slots
### Code

```



```

C
int calculateHeight( int dist, int height1, int height2 )
{
    int minH = min( height1, height2);
    int maxH = max( height1, height2 );

```

```

if( dist == 0 ) return 0;
if( dist == 1 ) return minH + 1;

// if both heights are same then meet in middle
if(minH == maxH )
{
    int add = ( dist%2 == 0 ) ? dist/2 : dist/2+1;
    return minH + add;
}

```

```
// See if we can make the height same
int delta = maxH-minH;
if( delta < dist )
{
    dist -= delta;
    minH += delta;
    int add = ( dist%2 == 0 ) ? dist/2 : dist/2+1;
    return minH+add;
}

// for all cases where delta >= dist
return minH+dist;
```

```
}
```

```
int getMaxHeight( const vector& positions, const vector& heights )
{
if( positions.empty() || heights.empty() || positions.size() != heights.size() )
{
return 0;
}
```

```
int result = 0;
for(int i=1; i<positions.size(); ++i )
{
    int currMax = calculateHeight( positions[i]-positions[i-1]-1, heights[i], height );
    result = max( result, currMax );
}
return result;
```

```
}
```

```
int main() {
cout << getMaxHeight({1,10}, {1,5}) << endl;
cout << getMaxHeight({1,3,7},{4,3,3}) << endl;
cout << getMaxHeight({1,2,4,7},{4,5,7,11}) << endl;
}
```

py

```
--  
## Twitter. Efficient Job Processing Service / 8/29  
  
  
01背包問題的變形  
[](https://leetcode.com/discuss/interview-question/374446/Twitter-or-OA-2019-or-Eff  
### 思路
```

```
### Code
```



```
py
```

```
"""
```

```
Complete the 'maximumTotalWeight' function below.
```

The function is expected to return an INTEGER.

The function accepts following parameters:

1. INTEGER_ARRAY weights
2. INTEGER_ARRAY tasks
3. INTEGER p

```
"""
```

```
def maximumTotalWeight(weights, tasks, p):  
    p = p  
    l = len(weights)  
    tasks = [2*i for i in tasks]  
    K = [[0 for i in range(p+1)] for j in range(l+1)]  
    for i in range(l+1):  
        for j in range(p+1):  
            if i == 0 or j == 0:  
                K[i][j] = 0  
            elif weights[i-1] <= j:  
                K[i][j] = max(tasks[i-1]+K[i-1][j-weights[i-1]], K[i-1][j])  
            else:  
                K[i][j] = K[i-1][j]  
            j = p  
            s = 0  
    result = K[l][p]  
    for i in range(l, 0, -1):  
        if result <= 0:  
            break  
        if result == K[i-1][j]:
```

```
        continue
    else:
        s += weights[i-1]
        print(weights[i-1], tasks[i-1])
        result -= tasks[i-1]
        j -= weights[i-1]
    return s
```

```
if name == 'main':
```

```
----  
## Twitter. Game Events / 9/20  
  
### 路線  
常規表示式的字串處理....
```

[](<https://docs.python.org/2/library/re.html>)

先創造一個output字串陣列

針對其中一個事件

用常規表示式加入規則 -> split_event

過濾出可以排序的參數 -> record

把record加入game_details_list然後繼續加入其他事件

依據給訂規則排序 game_details_list 並回傳正確順序的list[]

Code

```
py
import re

def getEventsOrder(team1, team2, events1, events2):
    # Write your code here
    football = list()
    football.append({"team": team1, "event": events1})
    football.append({"team": team2, "event": events2})
```

```
game_details_list = list()
original_event = list()
event_priority = ['G', 'Y', 'R', 'S']

for f in football:
```

```

for event in f["event"]:
    original_event.append(f["team"]+" "+event)

    # split events string to get details
    pattern = re.compile("( [a-zA-Z\s]*)(\d+)[+]?( \d*).([G,Y,R,S])( [a-zA-Z\s]*)")
    split_event = pattern.search(event)

    # create a list of format ["team name", "player name", "time", "extra time"]
    record = list()
    record.append(f["team"]) # team name
    if split_event:
        record.append(split_event.group(1).strip()) # player name
        record.append(int(split_event.group(2).strip())) # time
        record.append(int(split_event.group(3).strip()) if len(split_event.group(3)) > 0 else None) # extra time
        record.append(event_priority.index(split_event.group(4).strip())) # event priority
        record.append(split_event.group(5).strip()) # second player name
    game_details_list.append(record)

# sorting the list to return index position of the sorted list
new_num_index_sorted = (sorted(range(len(game_details_list))),
                        key=lambda k: (
                            game_details_list[k][2], # time
                            game_details_list[k][3], # extra time
                            game_details_list[k][4], # event
                            game_details_list[k][0], # team name
                            game_details_list[k][1], # player name
                            game_details_list[k][5])))

# based on the index position, fetching result from original event list and append it
answer = list()
for i in new_num_index_sorted:
    answer.append(original_event[i])
return answer

```



```

if name == 'main':
    team1 = "EDC"
    events1 = ['Name1 12 G', 'FirstName LastName 43 Y']
    team2 = "CDE"
    events2 = ['Name3 45+1 S SubName', 'Name4 46 G']
    getEventsOrder(team1, team2, events1, events2)

```



更正idTable裡的數之後，

要把舊的減少，新增加的增加

Code

```
py
from collections import Counter

def getUniqueIdUserIdSum(arr):
    idTable = Counter(arr)
    lowestNum = 1
    returnSum = 0
    for i in arr:
        if idTable[i] > 1:
            lowestNum = i + 1
        while idTable[lowestNum] > 0:
            lowestNum += 1
            idTable[lowestNum] -= 1
            idTable[i] -= 1
            returnSum += lowestNum
        else:
            returnSum += i
    return returnSum
```

```
if name == 'main':
    print(getUniqueIdUserIdSum([3,2,1,2,7]))
    print(getUniqueIdUserIdSum([1,1,1,1,1]))
    print(getUniqueIdUserIdSum([1,2,3,4,5]))
    print(getUniqueIdUserIdSum([1,100,1,1]))
    print(getUniqueIdUserIdSum([12,11,12,11]))
```

```
## Twitter. Partitioning array / 9/20

### 思路
```

1. make sure numbers.length is divisible by k
2. no element appears more than numbers.length/k times

Code

```
py
from collections import Counter

def solve(k, nums):
    if k == 0 or len(nums) == 0:
        return "YES"
    if len(nums)%k != 0:
        return "NO"
    counter = Counter(nums)
    for entry in counter:
        if counter[entry] > len(nums)/k:
            return "NO"
    return "YES"

if name == 'main':
    print(solve(2,[1,2,3,4]))
    print(solve(2,[1,2,3,3]))
    print(solve(3,[1,2,3,4]))
    print(solve(3,[3,3,3,6,6,6,9,9,9]))
    print(solve(1,[]))
    print(solve(1,[1]))
    print(solve(2,[1,2]))
```

```
---
## Twitter. Autoscale Policy / 9/21


[](https://leetcode.com/discuss/interview-question/376019/Twitter-or-OA-2019-or-Aut

### 技巧

- String format :
  - Python3
    - print(f'{i}-th second, utility is {A[i]}, instance number before action is {a
  - Python2
    - print('{}-th second, utility is {}, instance number before action is {}'.format(i, A[i], a))

### 思路

蠻簡單的問題

### Code
```

py

```
def autoScale(ins, A):
    lim_l, lim_h, th_l, th_h, idle = 1, int(1e8), 25, 60, 10
    i, n, ans = 0, len(A), ins
    while i < n: #print(f'{i}-th second, utility is {A[i]}, instance number before action is {ans}')
        print(f'{i}-th second, utility is {A[i]}, instance number before action is {ans}'.format(i, A[i], ans)) if A[i] < th_l and ans > lim_l:
            ans, i = (ans + 1) // 2, i + idle
        elif A[i] > th_h and ans <= lim_h:
            ans, i = ans * 2, i + idle
        i += 1
    return ans
```

```
ins, A = 1, [5, 10, 80]
print(autoScale(ins, A))
```

```
## Twitter. Authentication Tokens / 9/21


[](https://leetcode.com/discuss/interview-question/378237/Twitter-or-OA-2019-or-Aut
### 技巧

- Count-up 特定條件：
  - count = sum(1 for i in values.values() if i >= time)

### 思路

簡單題

### Code
```

```
py
def numberOfTokens(expiryLimit, commands):
# Write your code here
values = dict()
time = 0
```

```
for c in commands:
    # extraction values
    action = c[0]
    token_id = c[1]
    time = c[2]

    # set token
```

```

if action == 0:
    values[token_id] = expiryLimit + time

# reset token
elif action == 1:
    # check if token exists
    if token_id in values.keys():
        expiry_time = values.get(token_id)
        if expiry_time >= time:
            values[token_id] = time + expiryLimit

# counting values alive after reading all the values
count = sum(1 for i in values.values() if i >= time)
return count

```

532. K-diff Pairs in an Array / 9/21

Given an array of integers and an integer k, you need to find the number of unique

思路



py

"""

k > 0 -> subtract elem in key -> dic[find(elem+k)] > 0 ? count+1:count

[3,1,4,1,5], k=2

'1': 2

'3': 1

'4': 1

'5': 1

k = 0 -> return duplicate elem count/2

[1,1,2,2,2,3,3,3] -> 3: (1,1), (2,2), (3,3)

[1,1,1,1,1] -> 1: (1,1)

edge:

nums = [] or [1]

k < 0

"""

Code

```

py
from collections import Counter
class Solution(object):
    def findPairs(self, nums, k):
        """
:type nums: List[int]
:type k: int
:rtype: int
"""

if nums is None or len(nums) == 0 or len(nums) == 1 or k < 0 : return 0
dicNum = Counter(nums)
ret = 0
if k == 0:
    for key in dicNum:
        if dicNum[key] >= 2:
            ret += 1
return ret
else:
    for key in dicNum:
        if dicNum[key+k] > 0:
            ret += 1
return ret

```

```

---
## Twitter. Buying Show Tickets / 9/21


### 思路

```

Everyone before Alex will purchase the same amount as Alex or smaller, if their tic

Everyone after Alex will purchase the amount than Alex or smaller if their `number` i

Code

py

```

def waitingTime(tickets, p):
    if p > len(tickets) or p < 0: return -1
    total = tickets[p]
    for i in range(len(tickets)):
        # ppl in front of p
        if i < p:
            if tickets[i] < tickets[p]:
                total += tickets[i]
            else:
                total += tickets[p]
        # ppl behind p
        elif i > p:
            if tickets[i] < tickets[p]:
                total += tickets[i]
            else:
                total += tickets[p] - 1

```

```

    return total

```

```
if name == 'main':  
    a = waitingTime([2,6,3,4,5],0)  
    print(a)  
    a = waitingTime([2,6,3,4,5],1)  
    print(a)  
    a = waitingTime([2,6,3,4,5],2)  
    print(a)  
    a = waitingTime([2,6,3,4,5],3)  
    print(a)  
    a = waitingTime([2,6,3,4,5],4)  
    print(a)
```

```
a = waitingTime([5,5,2,3],0)  
print(a)  
a = waitingTime([5,5,2,3],1)  
print(a)  
a = waitingTime([5,5,2,3],2)  
print(a)  
a = waitingTime([5,5,2,3],3)  
print(a)
```

```
## Twitter. Weird Faculty / 9/22  

```

思路

- 一開始想說用二元搜尋，但發現有更簡單，但關鍵是要把0改成-1
- 解法：
 - 把0改成-1
 - 先把分界點設在index 0, left = 0, right = sum(v)
 - 向右移動分界點，直到 left > right

Code

```
py  
def weirdFaculty(v):  
n = len(v)  
# Convert all the zeros to -1 as the zero gives us the negative score of -1  
  
for i in range(n):  
if not v[i]:  
v[i] = -1
```

```

# Find the total sum

rightSum = sum(v)
leftSum = 0

# Find the point where left sum is greater than the right sum
for i in range(n):
    if leftSum > rightSum:
        return i
    leftSum += v[i]
    rightSum -= v[i]
return n

```

Akuna Capital OA / 9/23
選擇題：

! [](assets/markdown-img-paste-20190923201109492.png)
! [](assets/markdown-img-paste-20190923200222455.png)

程式題：

! [](assets/markdown-img-paste-20190923201442878.png)

BADF005D

DF005D to Decimal = 14614741

1+4+6+1+4+7+4+1 = 28

28 to Hex = 1C

1C == BA ? invalid: valid

思路

Code

py

'''

Input: BADF005D

DF005D to Decimal = 14614741

1+4+6+1+4+7+4+1 = 28

```
28 to Hex = 1C
```

```
1C == BA ? invalid: valid
```

```
"""
```

```
def isValid(line):
if (line == "" or len(line) != 8):
    return False
first2 = line[:2]
last6 = line[2:]
intLast6 = int(last6, 16) # Hex to int
print intLast6
```

```
sum = 0
while intLast6 > 0:
    if intLast6 < 10:
        sum += intLast6
        break
    sum += intLast6 % 10
    intLast6 //= 10

print sum
first2 = ("0x" + first2).upper()
last6 = (hex(sum)).upper()

print ("first2: {}", first2)
print ("last6: {}", last6)
print ("-----")
return first2 == last6
```

```
if name == "main":
print(isValid("BADF00D5"))
print(isValid("1CC0FfEE"))
print(isValid("F10235FF"))
```

```
---
## Twitter. Activate Fountain / 9/25

### 技巧
- 兩次sort的先後順序：**先sort 盡量達成的條件，再sort 必須達成的**
- 宣告二維list: **waterArea = [[0,0] for _ in range(n)]**
```

思路

* After observation of fountain range we can see if we find a fountain that covers

```

* don't cover till this garden point.
* Means;
* Fountain A (x,y)
* and Fountain B (z,m)
* <p>
* if(x<=z and y>=m) then x,y already covering garden till point B
* <p>
* But if Fountain C ( q,r)
* <p>
* if(y < r ) then our fountain (x,y) is not covering till this point, Hence we need
* We don't need to worry about the x to change, since whatever the case, our minimu
*
* Complexity: O(n*log(n)) / O(n)
* 1 2 3 4 5 6
* [2,1,2,1,2,4] n = 6
*
* [(1,3),(1,3),(1,5),(3,5),(3,6), (2,6)]
*
* [(1,5),(1,3),(1,3),(2,6),(3,6),(3,5)] l = 1 and r = 5; i=0 F=1
* [(1,5),(1,3),(1,3),(2,6),(3,6),(3,5)] l = 1 and r = 5; i=1 {(1,3) is in (1,5)}
* [(1,5),(1,3),(1,3),(2,6),(3,6),(3,5)] l = 1 and r = 5; i=2 {(1,3) is in (1,5)}
* [(1,5),(1,3),(1,3),(2,6),(3,6),(3,5)] l = 1 and r = 6; i=3 {(2,6) is not in (1,5)}
* [(1,5),(1,3),(1,3),(2,6),(3,6),(3,5)] l = 1 and r = 6; i=4 {(3,6) is in (1,6)}
* [(1,5),(1,3),(1,3),(2,6),(3,6),(3,5)] l = 1 and r = 6; i=5 {(3,5) is in (1,6)}

### Code

```



```

py
def activateFountains(fountain):
n = len(fountain)
waterArea = [[0,0] for _ in range(n)]

# calculate the most left and right this fountain can be
for i in range(1,n+1):
    waterArea[i-1][0] = max(i - fountain[i-1], 1) # left-most of waterArea
    waterArea[i-1][1] = min(i+ fountain[i-1], n) # right-most of waterArea

# two condition,
# 1. try to have most right
# 2. must have included the least one, cuz we will not update our left then
# so we should let our 'must do', become the last sort
waterArea.sort(key = lambda x: x[1],reverse = True)
waterArea.sort(key = lambda x: x[0])

# turn on the 'least left' but also with 'max right'
minLeft = waterArea[0][0]
maxRight= waterArea[0][1]
neededFountain = 1 # cuz we turn it on, we should count from 1

for i in range(1,n):

```

```

thisLeft = waterArea[i][0]
thisRight= waterArea[i][1]
if thisLeft <= minLeft and thisRight >= maxRight:
    continue
else:
    if thisRight > maxRight:
        maxRight = thisRight
        neededFountain += 1

return neededFountain

```

```

if name == 'main':
print(activateFountains([0,0,0,3,0,0,2,0,0]), 2)
print(activateFountains([2,1,2,1,2,4]), 2)
print(activateFountains([3,0,2,0,1,0]), 2)
print(activateFountains([3,0,1,0,1,0]), 2)
print(activateFountains([3,0,1,0,0,1]), 2)
print(activateFountains([2,0,2,0,1,0]), 2)
print(activateFountains([2,0,0,0,0]), 3)
print(activateFountains([0,0,0,0,0]), 5)
print(activateFountains([1,2,1]), 1)
print(activateFountains([0,1,0]), 1)

```

739. Daily Temperatures / 9/26

Given a list of daily temperatures T, return a list such that, for each day in the

For example, given the list of temperatures T = [73, 74, 75, 71, 69, 72, 76, 73], y

Note: The length of temperatures will be in the range [1, 30000]. Each temperature is an integer in the range [0, 100].

技巧

- Stack in python:
 - push: stack.insert(0, elem)
 - pop: stack.pop(0)
 - top: stack[0]

思路

用到了遞減stack，

把元素擺在裡面，直到遇到比他大的，

再一次讓裡面比較小的全部出去

情境：大哥進監獄，把門打開讓小弟全部出去，再自己進去

Code

```

py
def findWarmday(temperature):
    nextWarmday = [0 for _ in range(len(temperature))]
    stack = [] # stack for looking for next warmer day

    for i in range(len(temperature)):
        print stack
        while len(stack) != 0 and temperature[i] > temperature[stack[0]]:
            print("tem[i] > tem[top]", temperature[i], temperature[stack[0]])
            top = stack.pop(0)
            nextWarmday[top] = i - top
        stack.insert(0,i)

    return nextWarmday

if name == "main":
    print(findWarmday([73, 74, 75, 71, 69, 72, 76, 73]), [1, 1, 4, 2, 1, 1, 0, 0])

```

780. Reaching Point / 9/27

思路

直接看是否是整數倍就可了

用反推的做

目標是(3,5) 前一個狀態是 (3,2)

再前一個狀態是(1,2)

可以用減法，但是取餘數更威猛

2 % 5 = 2

5 % 2 = 1

if x < y 我們去縮小y

if x > y 我們去縮小x

用mod，我們可以一個式子就做完兩件事！

Code

TLE:

```
py
def ReachingPoint(sx, sy, tx, ty):
if sx > tx or sy > ty:
return False
if sx == tx and sy == ty:
return True
```

```
    if (ReachingPoint(sx+sy, sy, tx, ty)):
        return True
    else:
        return ReachingPoint(sx, sx+sy, tx, ty)
```

Smart – Recursive:

```
py
def ReachingPoint(sx, sy, tx, ty):
if sx > tx or sy > ty:
return False
if sx == tx and (ty-sy) % sx == 0:
return True
if sy == ty and (tx-sx) % sy == 0:
return True
```

```
    return ReachingPoint(sx, sy, tx%ty, ty%tx)
```

Smart – Iterative:

```
py
def ReachingPoint(sx, sy, tx, ty):
if sx > tx and sy > ty:
return False
while (tx >= sx && ty >= sy):
if (tx > ty):
tx %= ty
else:
ty %= tx
if tx == sx: return (ty-sy) % sx == 0
```

```
if ty == sy: return (tx-sx) % sy == 0  
return False;
```

```
---  
## 547. Friend Circles | 10/2 //TODO Union Find解法  
There are N students in a class. Some of them are friends, while some are not. Thei  
Given a N*N matrix M representing the friend relationship between students in the c  

```

思路

盲點：誤以為這題有n*n個人，其實只有n個人

對於某個人，遍歷其好友，然後再遍歷其好友的好友，那麼就能把屬於同一個朋友圈的人都遍歷一遍，同時標記出

還沒有學習並查集(Union Find)解法

Code

Recursive:

會超過遞迴限制



py

```
class Solution(object):  
    def findCircleNum(self, M):  
        """  
        :type M: List[List[int]]  
        :rtype: int  
        """  
  
        n = len(M)  
        visited = [False for _ in range(n)]  
        count = 0  
  
        for i in range(n):  
            if visited[i] == True:  
                continue  
            self.dfs(M, visited, i)  
            count += 1  
  
        return count
```

```
def dfs(self, M, visited, k):  
    visited[k] = True  
    for i in range(len(M)):  
        if M[i][k] != 1 and visited[k] == True:
```

```
        continue
    self.dfs(M, visited, i)
```

Recursive:

不會超過遞迴限制！！

```
py
class Solution(object):
def findCircleNum(self, M):
"""
:type M: List[List[int]]
:rtype: int
"""

def dfs(node):
visited.add(node)
for friend in xrange(len(M)):
if M[node][friend] and friend not in visited:
dfs(friend)
```

```
circle = 0
visited = set()
for node in xrange(len(M)):
    if node not in visited:
        dfs(node)
        circle += 1
return circle
```

Iterative:

用queue來做，思路差不多

```
c
class Solution {
public:
int findCircleNum(vector<vector<int>>& M) {
int n = M.size(), res = 0;
vector<bool> visited(n, false);
queue<int> q;
for (int i = 0; i < n; ++i) {
if (visited[i]) continue;
```

```

q.push(i);
while (!q.empty()) {
int t = q.front(); q.pop();
visited[t] = true;
for (int j = 0; j < n; ++j) {
if (!M[t][j] || visited[j]) continue;
q.push(j);
}
}
++res;
}
return res;
}
};

```

256. Paint House | 10/3

There are a row of n houses, each house can be painted with one of the three colors

The cost of painting each house with a certain color is represented by a $n \times 3$ cost

Note:

All costs are positive integers.

思路

這道題說有n個房子，每個房子可以用紅綠藍三種顏色刷，每個房子的用每種顏色刷的花費都不同，限制條件是相

$dp[i][j] = dp[i][j] + \min(dp[i - 1][(j + 1) \% 3], dp[i - 1][(j + 2) \% 3]);$

**這個也比較好理解，如果當前的房子要用紅色刷，那麼上一個房子只能用綠色或藍色來刷，那麼我們要求刷到當

Code



```

py
class Solution(object):
def minCost(self, costs):
"""
:type costs: List[List[int]]
:rtype: int
"""

if len(costs) == 0 or len(costs[0]) == 0:

```

```
return 0
dp = costs
n = len(costs)
for i in range(1,n):
    for j in range(3):
        dp[i][j] += min(dp[i-1][(j+1)%3], dp[i-1][(j+2)%3])
```

```
return min(dp[n-1][0],min(dp[n-1][1],dp[n-1][2]))
```

```
---
## Twitter. Parking Dilemma / 10/3

```

有n台車分別停在a, b, c, d四個位置，給定一個目標k台車必須可以被屋頂遮住，

求最小的k

思路

先排序，再用滑動視窗解

Code

```
py
def carParkingRoof(cars, k):
    result = float("inf")
```

```
n = len(cars)
# Sorting the array.
cars.sort()

# Find minimum value among
# all K size subarray.
for i in range(n - k + 1):
    result = int(min(result, cars[i + k - 1] - cars[i] + 1))
return result
```

```
---
## 647. Palindromic Substrings / 10/3
```

Given a **string**, your task is to **count** how many palindromic substrings in this **strin**

The substrings with different start indexes or end indexes are counted as different

```

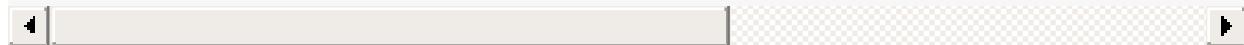
### 思路
```

以前找回文的方式：找到中心點後，

l指針向左， r指針向右，找尋其他可能的解

並且奇數回文、偶數回文都要考慮

Code



py

```
class Solution:
    def countSubstrings(self, s: str) -> int:
        if len(s) <= 1:
            return len(s)
```

```
count = 0
def checkLR(s, l, r):
    count = 0
    while l >= 0 and r < len(s) and s[l] == s[r]:
        count += 1
        l -= 1; r += 1
    return count

for i in range(len(s)):

    # odd case
    count += checkLR(s,i,i)

    # even case
    count += checkLR(s,i,i+1)

return count
```

```
---
```

```
## Twitter. SubPalindrome / 10/3
```

給個字符串

回傳他所有的回文字串，且不能重複

技巧

- 自動檢查有沒有重複：set()

```
- set.add()
```

思路

與前一提的基本回文找法一樣，並且使用set()去完成

Code

py

```
class Solution:
```

```
def countSubstrings(self, s: str) -> int:
```

```
if len(s) <= 1:
```

```
return len(s)
```

```
ret = set()
def checkLR(s, l, r, ret):
    count = 0
    while l >= 0 and r < len(s) and s[l] == s[r]:
        count += 1
        l -= 1; r += 1
    ret.add(s[l+1:r])

for i in range(len(s)):

    # odd case
    checkLR(s,i,i,ret)

    # even case
    checkLR(s,i,i+1,ret)

return ret
```

```
if name == "main":
```

```
s = Solution()
```

```
print(s.countSubstrings("aabaa"))
```

```
print(s.countSubstrings("aabbaa"))
```

Twitter. Restocking the warehouse / 10/3

思路

easy 就不做了

Twitter. Balanced Sales Array / 10/3

```
### 思路  
easy 也不做
```

```
## Twitter. University Careerfair / 10/3  

```

```
### 技巧
```

- 把兩個相關的元素綁在一起: `zip(list1, list2);` in our case: `(arrival, duration)`
- 兩次`sort`: `key=lambda p: (p[0]+p[1], p[1])`
 - `sort`的結果會是按照`p[0]+p[1]`去排，如果相同，再用`p[1]`去排 <- 這個做法比拆開來做直觀！要學！
 - 回想：兩次`sort`的先後順序性 -> 次要條件先做，主要條件後做
 - 所以要拆開來寫的話：
 - `key=lambda p: (p[1])`
 - `key=lambda p: (p[0]+p[1])`

```
### 思路
```

我在意的是：

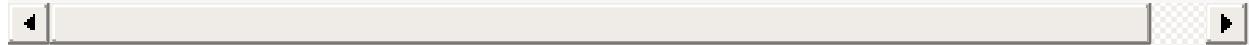
你們什麼時候離開？早離開的先排入

如果同時離開？duration短的先排入

```

```

```
### Code
```



```
py  
def universityCareerFair(arrival, duration):  
    aux = sorted(  
        list(zip(arrival, duration)), # bind same index in two list into a pair  
        key=lambda p: (p[0]+p[1], p[1]) # sort by 1. leave time 2. duration time  
    )  
    print(aux)  
    ans, end = 0, -float('inf')  
    for arr, dur in aux:  
        if arr >= end:  
            ans = ans + 1  
        end = arr + dur # 更新時間  
    return ans  
  
print(universityCareerFair([1, 3, 3, 5, 7], [2, 2, 1, 2, 1])) # 4  
print(universityCareerFair([1, 2], [7, 3])) # 1  
print(universityCareerFair([1, 3, 4, 6], [4, 3, 3, 2])) # 2
```

```
print(universityCareerFair([1, 2, 3, 4, 5], [3, 1, 1, 1, 1])) # 4
```

Twitter_Anagram | 10/3

Two words are anagrams of one another if their letters can be rearranged to form the other.

In this challenge, you will be given a **string**. You must split it into two contiguous substrings.

For example, given the **string** 'abccde', you would break it into two parts: 'abc' and 'cde'.

Function Description

Complete the anagram function in the editor below. It should return the minimum number of operations required to make the two substrings anagrams.

Anagram has the following parameter(s):

s: a **string**

Input Format

The first **line** will contain an **integer**, , the number of test cases.

Each test case will contain a **string** which will be concatenation of both the strings.

The given **string** will contain only characters in the range ascii[a-z].

Constraints

consists only of characters in the range ascii[a-z].

Output Format

For each test case, print an **integer** representing the minimum number of changes required to make the two substrings anagrams.

思路

統計成字典後就是分養樂多的概念：

你一瓶我一瓶，更快的做法是

所以我們不需要在意可以分成兩邊的養樂多，

我們只要在意那些"多出來的"，也就是奇數的

=> sum(each%2) / 2

結果我沒有搞清楚題目的意思，題目是說要連續的字串.....

我的作法是全部搞再一起 ==

```

```

技巧

- Convert a **string** to a list obj:
 - li = list(str)
- Access Dictionary by index:
 - Counter.most_common()[index]

- 存取字典裡不確定的存不存在的方法：
 - Counter.get(char, 0)
 - 回傳Counter[char]，如果不存在的話回傳0

Code
字元位置可以調換的：



```
py
from collections import Counter

def anagram(input):
if len(input)%2 != 0:
return -1
```

```
    li = list(input)
    dic = Counter(li)
    count = 0
    for i in range(len(dic)):
        count += (dic.most_common()[i][1]) % 2
    return count//2
```

```
if name == 'main':
```

```
print(anagram("aaaabbbcccd"))#1
print(anagram("xaxbbbxx"))#1
print(anagram("xxxbba"))#1
print(anagram("aaabbb")) #1
print(anagram("mnop")) #2
print(anagram("ab")) #1
print(anagram("xyyx")) #0
```

字元位置固定的：(符合提議的)

```
py
def anagram2(input):
s = input
l = len(s)
if l % 2 == 1:
return -1
```

```

else:
count = 0
s1, s2 = Counter(s[:l//2]), Counter(s[l//2:])
for char in s2:
current = s2[char] - s1.get(char,0)
if current > 0:
count += current
print(count)

```

```

---
## Flatiron_RenamePhoto | 10/5
輸入：一個超長字符串，包含換行，每行的格式一致為 name + city + date time

輸出：group by city, sort by date time，圖片重新命名為：city+index（按city groupby+時間順
其中，index的格式要 在前面適量的加0。比如說 這個City一共有10張照片，那麼index為1-9
按input順序 輸出重新命名的名字（合為一個大string輸出）
### 思路

```

1. 用正規表示式切割出元素
2. 依據地點、時間去排序
3. 加入到key為地點的字典中
4. 重新組裝字串
 - 需要處理補0的部分
 - 假設101，表示這個地點的index需要用到三位去存
 - 1-9 補兩個0，10-99補一個0，>100 不用補
 - 用兩個counter，一個是總位數，另一個是index i，總位數 - i//10，就可以知道這個index需要補

技巧

1. 正則表達式其實不複雜，不要搞得太複雜！
 - patter = re.compile("你要的pattern")
 - pattern 實在不難，就像是我例子中的那樣
 - split_pic = pattern.search(str)
 - 根據你pattern中的()次序找出你要的東西
 - 第一個括號的東西在：split_pic.group(1)

Code



```

py
import re
def rename(input):
# Jeff_Taichung_20190804.jpg
pattern = re.compile("[a-zA-Z]+[a-zA-Z]+(\d+).+")
detailList = list()
for pic in li:

```

```

#pic = re.sub('[^a-zA-Z0-9 \n.]', ' ', pic) # pic = Jeff Taichung 20190804.jpg
split_pic = pattern.search(pic)
picDetail = list()
if split_pic:
    picDetail.append(split_pic.group(2)) # Location
    picDetail.append(split_pic.group(3)) # Time
    picDetail.append(split_pic.group(1)) # Name
    picDetail.append(split_pic.group(4)) # format
    detailList.append(picDetail)

```

```

sortedDetailList = sorted(detailList, key=lambda k: (k[0],k[1]))

locDic = dict()
for entry in sortedDetailList:
    if entry[0] in locDic.keys():
        locDic[entry[0]].append(entry)
    else:
        locDic[entry[0]] = []
        locDic[entry[0]].append(entry)

out = []
for location in locDic:
    zero_counter = 0
    locCounter = len(locDic[location])
    while locCounter > 0:
        locCounter /= 10
        zero_counter += 1
    pic_counter = 0

    for pic in locDic[location]:
        pic_counter += 1
        num_zero = zero_counter - pic_counter//10
        str_zero = "0" * num_zero
        out.append(pic[0]+ "_" +str_zero+str(pic_counter)+"_" +pic[2]+ "." +pic[3])
print(out)

```

```

if name == "main":
    input = "Jeff_Taichung_20190804.jpg Ian_Taipei_20190801.jpg
    Eason_Taichung_20190802.png"
    # output: ['Taichung_01_Eason.png', 'Taichung_02_Jeff.jpg', 'Taipei_01_Ian.jpg']
    rename(input)

```

```

## 399. Evaluate Division(*****) | 10/7
Equations are given in the format A / B = k, where A and B are variables represente

```

Example:

```
Given a / b = 2.0, b / c = 3.0.
```

```
queries are: a / c = ?, b / a = ?, a / e = ?, a / a = ?, x / x = ? .
```

```
return [6.0, 0.5, -1.0, 1.0, -1.0].
```

```
The input is: vector<pair<string, string>> equations, vector<double>& values, vector<string> queries
```

```

```

思路

This code could be more concise.

Put every number into a two-layer dictionary to record it's multiply.

For each query pair, check whether they're in the dictionary or not.

If both exist => solution must exist.

Record the multiply and the number we've visited already, keep seeking the dictionary.

For example:

```
equation = [["a", "e"], ["b", "e"]], value = [4, 3], query = [[["a", "b"]]]
```

```
Construct a dictionary = {"a": {"a": 1, "e": 4}, "b": {"b": 1, "e": 3}, "e": {"a": 1/4, "b": 1/3}
```

```
Follow the path: dct["a"] => dct["e"] => dct["b"], we have query [[["a", "b"]]] = 4 x 3 = 12
```

Code



py

```
class Solution:
    def calcEquation(self, equations: List[List[str]], values: List[float], queries: List[List[str]]) ->
        List[float]:
```

```
        def dfs(a, b, visited):
            if b in mem[a]:
                return mem[a][b]
            for c in mem[a].keys():
                if c not in visited:
                    v = dfs(c, b, visited + [c])
                    if v != -1.0:
                        return mem[a][c] * v
            return -1.0
        mem = collections.defaultdict(dict)
        for i, pair in enumerate(equations):
            a, b, v = pair[0], pair[1], values[i]
```

```
    mem[a][b], mem[b][a] = v, 1/v
    mem[a][a], mem[b][b] = 1.0, 1.0
    return [dfs(q[0], q[1], []) if mem[q[0]] and mem[q[1]] else -1.0 for q in queri
```



py

```
class Solution(object):
    def calcEquation(self, equations, values, query):
        def seeker(a, b, path=[]):
            # seek the result of a/b
            if a not in dct.keys() or b not in dct.keys(): # No solution
                return 0
            if b in dct[a]: # This is it!
                return dct[a][b]
            else: # Keep looking for solution
                tmp = []
                for c in dct[a].keys():
                    if c not in path and (seeker(c, b, path+[c])):
                        return dct[a][c]*(seeker(c, b, path+[c]))
        return
```

```
dct = {}                                # Put every number into the dict
for i in xrange(len(equations)):
    nums = equations[i]
    div = float(values[i])
    if nums[0] in dct.keys():
        dct[nums[0]][nums[1]] = div
    else:
        dct[nums[0]] = {nums[0]:1, nums[1]:div}
    if nums[1] in dct.keys():
        dct[nums[1]][nums[0]] = 1.0/div
    else:
        dct[nums[1]] = {nums[1]:1, nums[0]:1.0/div}

res = []
for pair in query:                      # seek the solution
    if seeker(pair[0], pair[1]):
        res += seeker(pair[0], pair[1]),
    else:
        res += -1,
return [float(n) for n in res]
```

--
695. Max Area of Island | 10/8
Given a non-empty 2D array grid of 0's and 1's, an island is a group of 1's (repres

Find the maximum area of an island in the given 2D array. (If there is no island, t

思路

就是遍歷

注意可以用同個矩陣去省去visited的空間

Code



py

```
class Solution:  
    def maxAreaOfIsland(self, grid: List[List[int]]) -> int:  
        def findIsland(i, j, count):  
            if grid[i][j] != 1:  
                return count  
            else:  
                count += 1  
                grid[i][j] *= -1  
                for k in range(4):  
                    if 0 <= i+dx[k] < col and 0 <= j+dy[k] < row:  
                        count = findIsland(i+dx[k], j+dy[k], count)  
            return count
```

```
if len(grid) == 0 or len(grid[0]) == 0:  
    return 0  
dx = [1, -1, 0, 0]  
dy = [0, 0, 1, -1]  
row, col = len(grid[0]), len(grid)  
  
maxCount = 0  
for i in range(col):  
    for j in range(row):  
        if grid[i][j] == 1:  
            maxCount = max(maxCount, findIsland(i, j, 0))  
return maxCount
```

Iterative:

py

class Solution:

```
def maxAreaOfIsland(self, grid: List[List[int]]) -> int:
```

```
def bfs(i, j, count):
    queue = [(i,j)]
    grid[i][j] = 0
    while queue:
        i, j = queue.pop(0)
        count += 1
        for dx, dy in ((0,1), (1,0), (0,-1), (-1, 0)):
            nx, ny = i+dx, j+dy
            if 0 <= nx < len(grid) and 0 <= ny < len(grid[0]) and grid[nx][ny] == 1:
                grid[nx][ny] = 0
                queue.append((nx,ny))
    return count

res = 0
for i in range(len(grid)):
    for j in range(len(grid[0])):
        if grid[i][j] == 1:
            res = max(res, bfs(i,j, 0))
return res
```

短小精幹....

py

```
def maxAreaOfIsland(self, grid):
```

```
m, n = len(grid), len(grid[0])
```

```
def dfs(i, j):
    if 0 <= i < m and 0 <= j < n and grid[i][j]:
        grid[i][j] = 0
        return 1 + dfs(i - 1, j) + dfs(i, j + 1) + dfs(i + 1, j) + dfs(i, j - 1)
    return 0

areas = [dfs(i, j) for i in range(m) for j in range(n) if grid[i][j]]
return max(areas) if areas else 0
```

```

## ***[Start Wayfair pre-0A]***

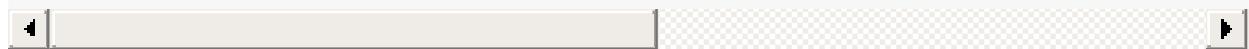
---

## 840. Magic Squares In Grid / 10/9
A 3 x 3 magic square is a 3 x 3 grid filled with distinct numbers from 1 to 9 such
Given an grid of integers, how many 3 x 3 "magic square" subgrids are there? (Each

### 思路

```

Code



py

```

class Solution:
    def numMagicSquaresInside(self, grid: List[List[int]]) -> int:
        def isValid(i,j):
            exist = set()
            # see if k is only from 1~9 and appear once
            for x in range(i,i+3):
                for y in range(j,j+3):
                    k = grid[x][y]
                    if k < 1 or k > 9 or k in exist:
                        return False
                    exist.add(k)
            return True

```

```

            if (15 != grid[i][j] + grid[i][j + 1] + grid[i][j + 2]): return False;
            if (15 != grid[i + 1][j] + grid[i + 1][j + 1] + grid[i + 1][j + 2]): return
            if (15 != grid[i + 2][j] + grid[i + 2][j + 1] + grid[i + 2][j + 2]): return
            if (15 != grid[i][j] + grid[i + 1][j] + grid[i + 2][j]): return False;
            if (15 != grid[i][j + 1] + grid[i + 1][j + 1] + grid[i + 2][j + 1]): return
            if (15 != grid[i][j + 2] + grid[i + 1][j + 2] + grid[i + 2][j + 2]): return
            if (15 != grid[i][j] + grid[i + 1][j + 1] + grid[i + 2][j + 2]): return Fal
            if (15 != grid[i + 2][j] + grid[i + 1][j + 1] + grid[i][j + 2]): return Fal
            return True;

```

```

count = 0
m, n = len(grid), len(grid[0])
for i in range(m-2):
    for j in range(n-2):
        if grid[i+1][j+1] == 5 and isValid(i,j):
            count += 1

return count

```

```
--  
## 443. String Compression / 10/11  
Given an array of characters, compress it in-place.  
  
The length after compression must always be smaller than or equal to the original a  
Every element of the array should be a character (not int) of length 1.  
  
After you are done modifying the input array in-place, return the new length of the  
  
Follow up:  
Could you solve it using only O(1) extra space?  
  
  
  
### 思路  
  
[a,b,b,b]  
  
[a,b,b]  
  
[a,b]  
  
[a,b,3]  
  
### Code
```

```
py  
class Solution(object):  
    def compress(self, chars):  
        if len(chars) == 0 or len(chars) == 1:  
            return len(chars)  
  
        i = 0  
        while i < len(chars)-1:  
            count = 1  
            while i < len(chars)-1 and chars[i] == chars[i+1]:  
                count += 1  
                del chars[i]  
  
            if count > 1:  
                count = str(count)  
                n = len(count)  
                for j in range(n):  
                    chars.insert(i+1+j, count[j]) #insert the number after the letter
```

```
i += n+1
else:
    i += 1 # directly move on

return len(chars)
```

```
--  
## Flatiron. CityNetwork / 10/11
```

給一個城市與城市連結的關係：

```
[9, 1, 4, 9, 0, 4, 8, 9, 0, 1]
```

index 為A城市，value為B城市

可以解讀成：0跟9相連、1跟1相連、2跟4相連....

定義首都為，city[i] = i

return 一個distance array

index為與首都的距離，value為距離首都此距離的城市個數

```
input: [9, 1, 4, 9, 0, 4, 8, 9, 0, 1]
output: [1, 1, 3, 2, 3, 0, 0, 0, 0]
```

思路

用queue去解

Code

py

```
from collections import deque
def network(city):
    res = [0] * (len(city) - 1)
    q = deque()
    for i in range(len(city)):
        if city[i] == i:
            q.append(i)
    res[0] = 1
```

```
dis = 1
while dis < len(city)-2:
    # 設定這次的目標城市(首都)
    num = []
```

```

while q:
    num = num + [q.pop()]

# 找出與此目標相連的城市(9)
count = 0
for i in range(len(num)):
    for j in range(len(city)):
        if city[j] == num[i] and j != num[i]:
            count += 1 # 計算這個距離的城市個數
            q.append(j) # 設定下一輪的目標 (9)

res[dis] = count
dis += 1
return res

```

```

if name == "main":
print(network([9,1,4,9,0,4,8,9,0,1]))

```

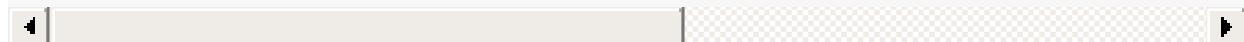
215. Kth Largest Element in an Array / 10/11
Find the kth largest element in an unsorted array. Note that it is the kth largest

```


### 技巧
- heap: import heapq
  - heapq.heapify(nums)
  - list = heapq.nlargest(k, list)

### Code

```



```

py
import heapq
class Solution:
def findKthLargest(self, nums: List[int], k: int) -> int:
heapq.heapify(nums)
return heapq.nlargest(k, nums)[-1]

```

Wayfair. Upsidedown the '6' and '9' | 10/11


```

## 技巧
python 不能支援string assignment
- astr[i] = "9"      <- Wrong
  - astr = astr[:i] + "9" + astr[i+1:]      <- Correct!

```

```
### 思路  
把第一個遇到的6換成9就可以了
```

```
### Code
```

```
py  
def upsidedown(A):  
astr = str(A)  
for i in range(len((astr))):  
if astr[i] == "6":  
astr = astr[:i] + '9' + astr[i+1:]  
return int(astr)  
return A  
  
if name == 'main':  
print(upsidedown(99666)) # 99966  
print(upsidedown(696)) # 996  
print(upsidedown(6999999999999999)) # 9999999999999999
```

```
---  
## Wayfair. Binary to zero / 10/11  

```

思路

第一個想法是轉成整數在弄成零，

但仔細觀察之後發現可以使用向右移的觀念。

遇到1的時候要先減一才能右移，零則可以直接移動。

最後一個leading 1只需要減一不需要右移

但要注意leading 1左邊的零是沒意義的所以要判斷掉

```
### Code
```

```
py  
from collections import Counter  
def steps(binary):  
lead = 0  
for i in range(len(binary)):  
if binary[i] == "1":  
lead = i  
break
```

```
if i == len(binary)-1: # input = 00000
    return 0
```

```
    li = list(binary[lead:])
    dic = Counter(li)
    return dic['1']*2 + dic['0'] - 1
```

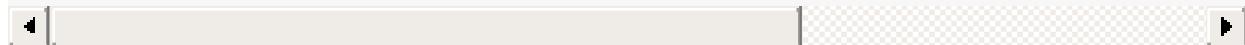
```
if name == "main":
    print(steps("010100"))
```

```
---
## 1189. Maximum Number of Balloons / 10/11
Given a string text, you want to use the characters of text to form as many instances of the word "balloon" as possible. Each character in text can only be used once. Return the maximum number of instances we can form.

You can use each character in text at most once. Return the maximum number of instances.


### 思路
```

```
### Code
```



```
py
from collections import Counter
class Solution:
    def maxNumberOfBalloons(self, text: str) -> int:
        dic = Counter(list(text))
        single = min(min(dic['b'], dic['a']), dic['n'])
        double = min(dic['l'], dic['o'])
        return min(single, double//2)
```

```
---
## 347. Top K Frequent Elements(Heap用法經典題) | 10/12
Given a non-empty array of integers, return the k most frequent elements.


```

```
### 技巧
```

- 一行建立字典的頻率跟數字的pair
 - mydic = [(freq, num) for num, freq in Counter(nums).items()]
- heapq.heappushpop()
- 遞減排序 sorted(arr, key=lambda k:{}), reverse=True)

```
- Counter(arr).most_common(k)
  - heapq.nlargest(k, arr)
```

思路

Code

真的練習到heap！不使用sort， $O(n)$

py

```
from collections import Counter
import heapq
```

class Solution:

```
def topKFrequent(self, nums: List[int], k: int) -> List[int]:
    mydic = [(freq, num) for num, freq in Counter(nums).items()]
    heap = []
    for entry in mydic:
        # always keep the most common k in heap by keeping the size of the heap
        if len(heap) >= k:
            heapq.heappushpop(heap, entry)
        else:
            heapq.heappush(heap, entry)
    res = [heap[i][1] for i in range(len(heap)) ]
    return res
```

$O(n \log n)$

py

```
from collections import Counter
class Solution:
    def topKFrequent(self, nums: List[int], k: int) -> List[int]:
        dict = Counter(nums)
        pair = []
        for entry in dict:
            pair.append([entry, dict[entry]])
```

```
        sortPair = sorted(pair, key=lambda n:n[1], reverse=True)
        res = []
        for i in range(k):
            res.append(sortPair[i][0])
```

```
    return res
```

O(n log n)

py

```
class Solution(object):
    def topKFrequent(self, nums, k):
        return [item[0] for item in collections.Counter(nums).most_common(k)]
```

239. Sliding Window Maximum(Deque經典題) / 10/12

Given an array nums, there is a sliding window of size k which is moving from the v

Note:

You may assume k is always valid, $1 \leq k \leq$ input array's size for non-empty array.

Follow up:

Could you solve it in linear time?

Hint:

How about using a data structure such as deque (double-ended queue)?

The queue size need not be the same as the window's size.

Remove redundant elements and the queue should store only elements that need to be

思路

Code



py

```
from collections import deque
class Solution:
    def maxSlidingWindow(self, nums, k):
        orderQueue = deque() # ordered index by importance
        res = []

```

```
        for i in range(len(nums)):
            # keep the importance ordered
            while orderQueue and nums[orderQueue[-1]] <= nums[i]:
```

```

        orderQueue.pop()

        # append the item
        orderQueue += [i]

        # keep the window size
        if i - orderQueue[0] >= k:
            orderQueue.popleft()

        # append the ans in to return list,
        # don't do this when still initialize the deque
        if i+1 >= k:
            res.append(nums[orderQueue[0]])

    return res

```

— — —

```

## Roblox. List Max / 10/14

### 思路



### Code
Naive:

```

```

py
def arrayManipulation(n, queries):
    arr = [0]*n
    for i in queries:
        for j in range(i[0], i[1] + 1):
            arr[j - 1] += i[2]
    return max(arr)

```

Best Solution:

```

py
def arrayManipulation(n, queries):
    arr = [0]*n
    for i in queries:
        arr[i[0] - 1] += i[2]
        if i[1] != len(arr):
            arr[i[1]] -= i[2]
    maxval= 0

```

```
itt = 0
print(arr)
for q in arr:
    itt += q
if itt > maxval:
    maxval = itt
return maxval
```

```
---
## Roblox. MostCommonSubstring | 10/14

### 技巧

- SubString in python: str[start, start+length]
- Counts of distinct char in str: len(set(str))
- Most Frequent var in dic: Counter().most_common(1)
```

```
### 思路
```

```
### Code
```

```
py
from collections import Counter
```

```
def MostCommonSubstring(str1, nl, xl, xu):
    dic = Counter()
    n = len(str1)
    for i in range(n):
        # i: start pos of substr
        if nl < n-i:
            maxB = xl
        else:
            maxB = n-1
```

```
        # j: length of substring
        for j in range(nl, maxB+1):
            if i+j <= n:
                substring = str1[i:i+j]
                distinctCount = len(set(substring))

                if len(substring) >= nl and len(substring) <= xl and distinctCount <= x:
                    if substring in dic.keys():
                        dic[substring] += 1
                    else:
```

```
        dic[substring] = 1
print(dic)
return dic.most_common(1)[0][1]
```

```
if name == "main":
print(MostCommonSubstring("abcde", 2, 4, 3))
print(MostCommonSubstring("abcde", 2, 4, 26))
print(MostCommonSubstring("ababab", 2, 3, 4))
```

```
---
## Roblox. Maximum Substring / 10/14
For this challenge, a substring is defined as any contiguous group of one or more c
```

思路

直接找出所有最大的char的位置

加進去candidate list

再排序一次

Code

```
py
def maxSubstring(s):
mxChar = sorted(s, reverse=True)[0]
cand = []
for i in range(len(s)):
if s[i] == mxChar:
cand.append(s[i:])
return(sorted(cand, reverse=True)[0])

if name == "main":
maxSubstring("cacbcabc")
maxSubstring("baca")
```

```
---
## Roblox. Segment | 10/14
```

1. 紿一個數組arr和一個數x，找出所有長度為x的subarray的最小值，返回這些最小值中的最大值
 1. arr = [8, 2, 4, 3]
 2. x = 2

3. 那麼subarray就是[8,2]和[2,4]和[4,3]，對應的最小值是2, 2, 3，那麼返回值就是3

思路
Code

```
py
def segment(arr, x):
cand = []
for i in range(len(arr)-x+1):
cand.append(arr[i:i+x])
```

```
minCand = []
for can in cand:
    minCand.append(min(can))
return max(minCand)
```

```
if name == "main":
print(segment([8,2,4,3],2))
```

Roblox. Repeated Word / 10/14

找一句話裡第一個重複單詞， 比如 "We work hard because hard work pays"， 返回hard，因為是第一
思路
Code

```
py
def RepeatedWord(str):
dic = dict()
li = str.split(" ")
for l in li:
if l not in dic.keys():
dic[l] = l
else:
return l

if name == "main":
print(RepeatedWord("We work hard because hard work pays")) # hard
print(RepeatedWord("We work Hard because hard work pays")) # ward
```

Break Palindrome / 10/14

給一個palindrome 問怎麼變可以讓palindrome變成非palindrome 並且lexicoorder最小

技巧

- increment char: ch = chr(ord(str[i]) + 1)
- 確認是否是回文：
 - isPalind(str): return str == str[::-1]

路線

- 更改第一個字元即可，但還是多做了一些防無限迴圈的處理

Code

py

```
def breakPalind(str):  
    i = 0  
    while isPalind(str):  
        ch = chr(ord(str[i]) + 1)  
        str = ch + str[1:]  
        if not isPalind(str):  
            return str  
        elif ch == 'z':  
            i += 1  
        if i >= len(str):  
            return "Unable to find the way to Break Palind"
```

```
def isPalind(str):  
    return str == str[::-1]
```

```
if name == "main":
```

```
    print(breakPalind("aabbaa"))  
    print(breakPalind("aabaa"))
```

Roblox. Prefix to Postfix Conversion / 10/14

Prefix : An expression is called the prefix expression if the operator appears in the beginning of the expression.
Example : *+AB-CD (Infix : (A+B) * (C-D))

Postfix: An expression is called the postfix expression if the operator appears in the end of the expression.
Example : AB+CD-* (Infix : (A+B * (C-D))

Given a Prefix expression, [convert](#) it into a Postfix expression.

路線

不用先轉成Infix，效率比較好

- Algorithm for Prefix **to** Postfix:
 - Read the Prefix expression in reverse order (**from right to left**)
 - If the symbol is an operand, then push it onto the Stack
 - If the symbol is an operator, then pop **two** operands **from** the Stack
 - Create a string by concatenating the **two** operands and the operator after them
 - string = operand1 + operand2 + operator
 - And push the resultant string back **to** Stack
 - Repeat the above steps until end **of Prefix expression.**

Code



```
py
def prefix2postfix(str):
    stack = []
    operator = ['+', '-', '*', '/']
    for i in range(len(str)-1, -1, -1):
        if str[i] in operator:
            if stack:
                operand1 = stack.pop()
            else:
                print("cannot get op1");break
            if stack:
                operand2 = stack.pop()
            else:
                print("cannot get op2");break
```

```
        stack.append(operand1+operand2+str[i])
    else:
        stack.append(str[i])
    if len(stack) == 1:
        return stack[0]
    else:
        print("stack size error")
```

```
if name == 'main':
    print(prefix2postfix("+AB-CD")) # AB+CD-
    print(prefix2postfix("-A/BC-/AKL")) # ABC/-AK/L-
```

```
follow-up
### 思路

### Code
```

```
py
def postfix2prefix(str):
    stack = []
    operator = ['+', '-', '*', '/']
    for i in range(len(str)):
        if str[i] in operator:
            if stack:
                operand1 = stack.pop()
            else:
                print("cannot get op1");break
            if stack:
                operand2 = stack.pop()
            else:
                print("cannot get op2");break
```

```
        stack.append(str[i] + operand2 + operand1)
    else:
        stack.append(str[i])
    if len(stack) == 1:
        return stack[0]
    else:
        print("stack size error")
```

```
if name == 'main':
print(postfix2prefix("AB+CD-")) #+AB-CD
print(postfix2prefix("ABC/-AK/L-*")) # *-A/BC-/AKL
```

```
---
## Pre/Post fix to Infix / 10/14
### 思路
### Code
```

```
py
def prefix2infix(str):
    stack = []
    operator = ['+', '-', '*', '/']
```

```
for i in range(len(str)-1,-1,-1):
if str[i] in operator:
if stack:
operand1 = stack.pop()
else:
print("cannot get op1");break
if stack:
operand2 = stack.pop()
else:
print("cannot get op2");break
```

```
stack.append(operand1 + str[i] + operand2)
else:
stack.append(str[i])
if len(stack) == 1:
return stack[0]
else:
print("stack size error")
```

```
def postfix2infix(str):
stack = []
operator = ['+', '-', '*', '/']
for i in range(len(str)):
if str[i] in operator:
if stack:
operand1 = stack.pop()
else:
print("cannot get op1");break
if stack:
operand2 = stack.pop()
else:
print("cannot get op2");break
```

```
stack.append(operand2 + str[i] + operand1)
else:
stack.append(str[i])
if len(stack) == 1:
return stack[0]
else:
print("stack size error")
```

```
if name == 'main':
```

```
print(prefix2infix("+AB-CD")) #((A+B)(C-D))
print(postfix2infix("AB+CD-")) #((A+B)(C-D))
```

```
---
## 總結Pre/Post/In Fix 轉換

- Prefix: *+AB-CD
- Postfix: AB+CD-*
- Infix: A+B * C-D

### 共同點:
- 都可用stack實現
- 遇到運算子就推進stack
- 遇到運算元就Pop兩個運算子出來{做點事}，再推回去stack

### 不同點:
- 從哪個方向讀起:
  - 從有運算子的方向讀
- 合併運算子及運算元(做點事):
  - 轉Infix:
    - pre2in:
      - op1 + op + op2
    - pos2in:
      - op2 + op + op1
  - Pre/Post直接轉換:
    - pre2post:
      - op1 + op2 + op
    - post2pre:
      - op2 + op1 + op
  - 小節:
    - **不要死記，看著結果順序去思考**:
      - **哪個會先推進stack**
      - **輸出又應該是要長怎樣**
```

```
## Roblox. Number of subarrays w/ m odd numbers | 10/14
Given an array of n elements and an integer m, we need to write a program to find t
```

```

```

```
### 技巧

- inline condition assign(Ternary Operator):
  - total_so_far = m==0 ? 0: 1
  - => total_so_far = 0 if m == 0 else 1
```

```
### 思路
- Two pointer 去列舉所有可能解的base
  - 再從base去列舉所有可能解
- start只有遇到超過m個odd的時候需要移動
```

```

```

```
- another solution: https://www.geeksforgeeks.org/number-subarrays-m-odd-numbers/
```

```
### Code
```

```
py
def getSubarrayCount(a, m):
    start,end = 0, 0
    cnt = 0 # cnt for odd m
    total = 0
    while end < len(a):
```

```
        if a[end] % 2 != 0:
            cnt += 1
        end += 1
        while cnt > m:
            if a[start] % 2 != 0:
                cnt -= 1
            start += 1
#        print("start = ", start);print("end = ", end);print("array: ", a[start:end]);p
        if cnt == m:
            #print("in coming!")
            total += getSubcount(a, start, end, m)
            #print(total);print("----")
    return total
```

```
def getSubcount(a, start, end, m):
```

```
    total_so_far = 0 if m == 0 else 1
    while start<end and a[start] %2 == 0:
        total_so_far += 1
        start += 1
    return total_so_far
```

```
if name == 'main':
```

```
    print(getSubarrayCount([2,5,6,9], 2))
    print(getSubarrayCount([2,2,5,6,9,2,11], 2))
```

```
----
## Roblox. Distinct Pairs / 10/14

```

```
## 技巧
- get the index of specific item in list:
    - list.index(item,{start}, {end})
```

```
### 思路  
### Code
```

```
py  
def distinctpair(arr, k):  
    res = set()  
    for elem in arr:  
        if elem > k:  
            continue  
        if k - elem in arr:  
            if k - elem > elem:  
                res.add((elem, k - elem))  
            else:  
                res.add((k - elem, elem))  
    return len(res)  
  
if name == 'main':  
    print(distinctpair([5,7,9,13,11,6,6,3,3],12))
```

```
---  
## Roblox. Social Network / 10/14  

```

```
### 技巧
```

- 不要輕易用int當作key，先轉成str
- 如果要用二維list
 - 先初始化: dic[size] = []
 - 再append: dic[size].append([i])

```
### 思路
```

```
### Code
```

```
py  
def socialnetwork(arr):  
    dic = dict()  
    out = []  
    for i in range(len(arr)):  
        size = str(arr[i])  
        if size in dic.keys():  
            # 如果該group已滿，重新創一個group  
            if len(dic[size][-1]) >= arr[i]:
```

```
dic[size].append([i])
else:
dic[size][-1].append(i)
else:
dic[size] = [] # 二維list初始化
dic[size].append([i])
```

```
# 如果group已滿，加入輸出list中
if len(dic[size][-1]) >= arr[i]:
    out.append(dic[size][-1])

print(sorted(out))
```

```
if name == 'main':
socialnetwork([2,1,1,2,1]) # 0 3 /n 1 /n 2 /n 4
print("----")
socialnetwork([3,3,3,3,3,1,3]) # 0 1 2 /n 3 4 6 /n 5
print("----")
socialnetwork([2,2,1,2,2]) # 0 1 /n 3 4 /n 2
```

```
---
## Google. Largest Subarray Length K / 10/17

```

follow-up: 如果數組可以重複
思路

不能重複就是找前k個的最大值，並以他為起點向後推k個回傳

如果數字可以重複就只能土法煉鋼幹

Code

py

'''

<https://leetcode.com/discuss/interview-question/352459/>

if every elem in A is distinct
if there is duplicate

'''

```
def largest_subarray_nodup(A, k):
n = len(A)
if n == 0 or k <= 0 or k > n:
```

```

return {}

if k == n:
    return A

cand = A[:n-k+2]
start = A.index(max(cand))
end = start + k
print(A[start:start+k])

def largest_subarray_wtdup(A, k):
    n = len(A)
    if n == 0 or k <= 0 or k > n:
        return {}
    if k == n:
        return A

    idx, max, = -1, 0
    i = 0

    while i <= n-k: if a[i]> max:
        max, idx = A[i], i
    elif A[i] == max:
        # comparing the rest, see if there is bigger elem in subarr start from A[i]
        newidx = i
        oldidx = idx
        while newidx < i+k: if A[newidx] > A[oldidx]:
            idx = i
            break
        newidx, oldidx = newidx+1, oldidx+1
        i += 1
    print(A[idx:idx+k])

if name == 'main':
    largest_subarray_nodup([1,4,3,2,5,1], 4) # [4,3,2,5]
    largest_subarray_wtdup([1, 4, 3, 2, 1, 4, 5, 6, 7],4) # [4,5,6,7]
    largest_subarray_wtdup([8,7,6,5,4,3,2,1],4) # [8,7,6,5]
    largest_subarray_wtdup([1,1,1,1,1,1],4)

```

Google. Maximum Time / 10/17

思路

Code

```

py
def maximum_time(time):
if time[0] == "2" or time[1] <= "3" or (time[0] == "?" and time[1] == "?"):
ret = "23:59"
else:
ret = "19:59"
for i in range(len(time)):
if time[i] != "?":
ret = ret[:i] + time[i] + ret[i+1:]
print ret

if name == 'main':
maximum_time("23:5?");# 23:59
maximum_time("2?:22");# 23:22
maximum_time("0?:??");# 09:59
maximum_time("1?:??");# 19:59
maximum_time("?:??");# 14:59
maximum_time("?:??");# 23:59!
maximum_time("?:??");# 23:59
maximum_time("?:5?"); #14:59
maximum_time("?:??"); #14:59
maximum_time("23:5?"); #23:59
maximum_time("2?:22"); #23:22
maximum_time("0?:??"); #09:59
maximum_time("1?:??"); #19:59
maximum_time("?:0?"); #14:09
maximum_time("?:4?"); #19:49

```

```

---
##### **Review Tree**
---

## 96. Unique Binary Search Trees / 10/18
Given n, how many structurally unique BST's (binary search trees) that store values


### 思路
The problem is to get the number of unique structurally BST that stores 1...n.

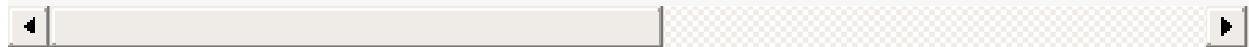
If n = 1, there is only one BST.

If n > 1, each r for 1 <= r <= n can be root. Then the problem is divided into two
How many structurally unique BST's that store values 1 ... r-1? (subproblem overlap)
How many structurally unique BST's that store values r+1 ... n?

```

The answer is the same as How many structurally unique BST's that store values 1 ..

If we define
result[i] as number of unique structurally BST that stores 1...i
then,
result[i] = sum(result[r-1]* result[i-r]) for `1 <= r <= n`
base case result[0] = 1 which represents an empty tree
Code



```
py
class Solution(object):
def numTrees(self, n):
"""
:type n: int
:rtype: int
"""
if n == 1: return 1
state = [0] * (n+1) # we want state[n] as the answer, so initial to N+1
```

```
    state[0] = 1 # initial
    for i in range(1, n+1):
        for r in range(1, i+1):
            state[i] += state[r-1] * state[i-r]

    return state[n]
```

--
105. Construct Binary Tree from Preorder and Inorder Traversal / 10/18
Given preorder and inorder traversal of a tree, construct the binary tree.

Note:

You may assume that duplicates do not exist in the tree.

For example, given

```
preorder = [3,9,20,15,7]
inorder = [9,3,15,20,7]

### 思路
```

解題方向：

找出root，找出左右子樹

1. preorder的頭一定是root
2. inorder中root以左是左子樹，以右是右子數

注意：

在找preorder的頭的時候用pop(0)，用[0]遇到空子樹就會gg

傳入inorder的時候也要跳過root (+1)

Code

py

```
class Solution(object):
    def buildTree(self, preorder, inorder):
        """
        :type preorder: List[int]
        :type inorder: List[int]
        :rtype: TreeNode
        """

        if preorder and inorder:
            root_idx_in_inorder = inorder.index(preorder.pop(0)) # 如果用preorder[0] 在遇到空子樹時會出錯
            root = TreeNode(inorder[root_idx_in_inorder])

            root.left = self.buildTree(preorder[:root_idx_in_inorder], inorder[:root_idx_in_inorder])
            root.right = self.buildTree(preorder[root_idx_in_inorder:], inorder[root_idx_in_inorder:])
            return root
```

Google. Compare Strings / 10/19 // **TODO:** More Testcase NEEDED

技巧

- 字串計算某字母出現的次數：
 - str.count()
- 回傳此字串中最小的字母
 - min(str)

```
### 思路
```

同個概念三種寫法

1. 徒法煉鋼
2. 漂亮解法
3. cache優化法

```
### Code
```

徒法煉鋼法：

py

```
def compare_string(A, B):
```

```
freq_count_of_A = [0] * 11 # count of freq 1~10
res = []
cache = [0] * 11 # cache for sum of first i value in freq_count_of_A
A = A.split(",")
for str in A:
    count = get_smallest_count(str)
    freq_count_of_A[count] += 1

B = B.split(",")
for str in B:
    count = get_smallest_count(str)

    if cache[count] == 0:
        sum = 0
        for i in range(1, count):
            sum += freq_count_of_A[i]
        cache[count] = sum
    else:
        sum = cache[count]

    res.append(sum)
return res
```

```
def get_smallest_count(str):
```

```
str = sorted(str)
i, count = 1, 1
while i < len(str) and str[i] == str[i-1]:
    count, i = count+1, i+1
```

```
return count
```

```
if name == 'main':  
    print(compare_string("abcd,aabc,bd", "aaa,aa"))
```

漂亮的解法

```
py  
def solve(A, B):  
    wordsA = A.split(",")  
    wordsB = B.split(",")  
    freqCounter = [0] * 11
```

```
for w in wordsA:  
    minFreq = w.count(min(w))  
    freqCounter[minFreq] += 1  
  
    toReturn = []  
    for w in wordsB:  
        minFreq = w.count(min(w))  
        toReturn.append(sum(freqCounter[:minFreq]))  
  
    return toReturn
```

漂亮的解法+cache優化

```
py  
def better_compare_string(A, B):  
    A, B = A.split(","), B.split(",")  
    freq_count_of_A, cache = [0] * 11, [0] * 11 # count of freq 1~10, cache for sum of first i value in  
    freq array  
    ret = []  
  
    for str in A:  
        min_freq = str.count(min(str))  
        freq_count_of_A[min_freq] += 1
```

```
    for str in B:  
        min_freq = str.count(min(str))  
        if cache[min_freq] == 0:  
            sum_f = sum(freq_count_of_A[:min_freq])  
            ret.append(sum_f)  
            cache[min_freq] = sum_f  
        else:
```

```
    ret.append(cache[min_freq])  
  
return ret
```

Google. Sum Of Leaves / 10/19 // **TODO:** Morris Traversal!

sum of leaf nodes of a binary tree

先寫出recursive DFS，然後問了iterative解法。寫出之後又問兩個解法的space complexity並要求寫出

思路

Follow-up: 查查Morris traversal，差不多就是那個思路，traverse下去的時候要把left改成parent

Code

Recursive:



```
py  
def add_leaves(root):  
if not root:  
    return 0  
if not root.left and not root.right:  
    return root.val  
    return add_leaves(root.left) + add_leaves(root.right)  
  
if not root:  
    return 0  
    return add_leaves(root)
```

Iterative:

```
py  
def iter_add_leaves(root):  
stack = []  
stack.append(root)  
val = 0  
while stack:  
    top = stack.pop()  
    if not top.left and not top.right:  
        val += top.val
```

```
if top.left:
    stack.append(top.left)
if top.right:
    stack.append(top.right)
return val
```

404. Sum of Left Leaves / 10/19
Find the sum of all left leaves in a given binary tree.

```

```

路徑

Recursive: 用一個flag去標示是否為左子樹

Iterative: 查看這個節點的未來狀態

Code

Recursive:

py

```
def sumOfLeftLeaves(self, root: TreeNode) -> int:
def add_left_leaves(root, isLeft):
if not root:
return 0
if not root.left and not root.right:
#print(root.val)
if isLeft:
return root.val
else:
return 0
```

```
        return add_left_leaves(root.left, True) + add_left_leaves(root.right, False)

if not root:    return 0
val = 0
return add_left_leaves(root.left, True) + add_left_leaves(root.right, False)
```

Iterative:

```
py
def iter_add_left_leaves(root):
    stack = []
    stack.append(root)
    val = 0
    while stack:
        top = stack.pop()
        if top.left:
            stack.append(top.left)
        if not top.left.left and not top.left.right:
            val += top.left.val
```

```
    if top.right:
        stack.append(top.right)
    return val
```

Google. Sum of Sliding Window / 10/19

Follow-up: Max of Sliding window

詳情請見 Leetcode 239

技巧

記得也要把slice所造成的時間複雜度算進去！

```
py
for i in range(len(nums)-k+1):
    res.append(sum(nums[i:i+k]))
```

```
### Code
Nearly brute force:
**竟然是O(n^2)!!**
因為slice的複雜度是O(k)
```

```
py
def sumSlidingWindow(self, nums, k):
    res = []
    for i in range(len(nums)-k+1):
        res.append(sum(nums[i:i+k]))
```

```
print(res)
```

Better way ([prefix sum](#)):

```
py
def sumSlidingWindow2(self, nums, k):
    curSum = sum(nums[:k])
    res = [curSum]
    stack = [i for i in nums[:k]]
    print(stack)
    for i in range(k, len(nums)):
        num = stack.pop(0)
        curSum = curSum - num + nums[i]
        res.append(curSum)
        stack.append(nums[i])
    print(res)
```

Google. Delete Alternate Linked List / 10/19 // Solved: 為什麼可以直接return head ,
刪除節點與節點中間的

<https://www.geeksforgeeks.org/delete-alternate-nodes-of-a-linked-list/>

思路

Keep track of previous of the node to be deleted.

當你在修改prev.next的時候就等於在修改head.next

Code

```
py
def delAlternate(self, head):
    # given 1->2->3->4->5, return 1->3->5
    # given 1->2->3->4 return 1->3
    prev = head
    node = head.next # the node is the one to be deleted
    while prev and node:
        prev.next = node.next
        # free(node)
        node = node.next
```

```
prev = prev.next
if prev:
    node = prev.next
return head
```

```
py
def recur_delAlternate(self, head):
    if not head:
        return
    node = head.next
```

```
    if not node:
        return
    head.next = node.next
    # free node
    self.recur_delAlternate(head.next)
    return head
```

Google. Locate Repeated 3 times or more word / 10/20

給一個string: abbbcccc, 標記出 連續重複三次或以上的 字母的初始位置和其實位置;

input: abbbcccc, output: [(1, 3), (4, 7)]

follow-up: 細出 一個 string 例如 hellllloooo, 求是否能通過remove 重複的字符得到 一個va

思路

valid english word可以用trie解

<https://www.geeksforgeeks.org/trie-insert-and-search/>

所有的合法單詞先構建一個Trie, 通過第一問的function可以獲得重複字母的區間，區間內的所有字符均可以

Code

```
py
def repeated3ormore(str):
    ret, i = [], 1
    while i < len(str): if str[i] == str[i-1]: start = i-1 count = 1 while i < len(str) and str[i] == str[i-1]:
```

```
count, i = count+1, i+1 end = i-1 if count >= 3:  
ret.append((start, end))  
else:  
i += 1  
return ret
```

```
if name == 'main':  
print(repeated3ormore("abbbcccc"))  
print(repeated3ormore("a"))  
print(repeated3ormore("bbb"))  
print(repeated3ormore("abbcccccbbc"))  
print(repeated3ormore(""))
```

follow-up, Sample code:

```
py  
def dfs(self, s, lis, dic):  
self.dic = dic  
s = list(s)  
def recur(s, lis):  
if ''.join(s) in dic:  
return True  
if lis == []:  
return False  
start, end = lis.pop()  
for i in range(start, end):  
s.pop(start)  
if recur(s, lis):  
return True  
return False
```

```
return recur(s, lis)
```

```
if name == 'main':  
lis = repeated3ormore("heelllloo")  
dic = {"hello"}  
dfs("heelllloo", lis, dic)
```

```
## 167. Two Sum II - Input array is sorted / 10/21
```

Given an array of integers that is already sorted in ascending order, find `twoSum`

The function `twoSum` should return indices of the two numbers such that they add up

Note:

Your returned answers (both `index1` and `index2`) are not zero-based.

You may assume that each input would have exactly one solution and you may not use

Example:

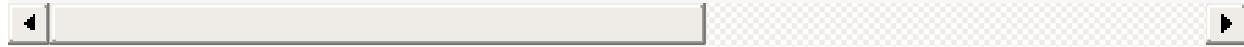
Input: `numbers = [2,7,11,15]`, `target = 9`

Output: `[1,2]`

Explanation: The sum of 2 and 7 is 9. Therefore `index1 = 1`, `index2 = 2`.

思路

Code



py

```
def twoSum(self, numbers: List[int], target: int) -> List[int]:  
    n = len(numbers)  
    start, end = 0, n-1  
    while end > start:  
        if numbers[end] + numbers[start] > target:  
            end -= 1  
        elif numbers[end] + numbers[start] < target:  
            start += 1  
        else:  
            return [start+1, end+1]  
    return [-1,-1]
```

```
## Remove the end node from list / 10/23
```

思路

Code

py

```
def removeEnd(self, head):
```

```
if not head or not head.next: return None
prev = ListNode(-1)
prev.next = head
while head and head.next:
if not head.next.next:
head.next = None
else:
head = head.next
return prev.next
```

19. Remove Nth Node From End of List | 10/21
Given a linked list, remove the n-th node from the end of list and return its head.

Example:

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

After removing the second node from the end, the linked list becomes 1->2->3->5.

Note:

Given n will always be valid.

Follow up:

Could you do this in one pass?

思路

只要會刪除到頭節點，

永遠要用一個duminode指著頭節點，

回傳就回傳duminode.next

須考慮這些測資：

[1] 1

[1,2] 2

[1,2,3] 3

follow-up:

one-path的方法是先讓快指針走了n步

在開始同時讓快慢指針同時前進

當快指針走到底時，慢指針會剛好指導倒數第n個 linklist[-n]

再將此節點刪除即可

一樣需要用到dummy的觀念

follow-up-up

不使用dummy node.....

等於要自己想一個方式處理刪除頭節點的情況

Code



py

class Solution:

def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:

if not head: return head

```
size = self.listsize(head)
target = size - n

dum = ListNode(-1) # 只要會刪除到頭節點，就要用一個duminode指向頭節點
dum.next = head
prev = dum
while head:
    if target != 0:
        prev = head # move prev
        head = head.next # move cur node
    else:
        prev.next = head.next # delete
        break
    target -= 1
return dum.next

def listszie(self, head):
    count = 0
    while head:
        count += 1
        head = head.next
    return count
```

one-path:

py

def removeNthFromEnd(self, head, n):

dummy = ListNode(0)

```
dummy.next = head
fast = slow = dummy
for _ in xrange(n):
    fast = fast.next
    while fast and fast.next:
        fast = fast.next
        slow = slow.next
        slow.next = slow.next.next
return dummy.next
```

one-path + space(1)

```
py
class Solution(object):
    def removeNthFromEnd(self, head, n):
        fast, slow = head, head
        for i in range(n):
            fast = fast.next
```

```
        if not fast: # deal with head deleting
            return slow.next

        while fast.next:
            fast = fast.next
            slow = slow.next

        slow.next = slow.next.next

        return head
```

24. Swap Nodes in Pairs | 10/22 // TODO: Figure out recursive way
Given a linked list, swap every two adjacent nodes and return its head.

You may not modify the values in the list's nodes, only nodes itself may be changed

Example:

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

Linked list 修改的總結

1. 先寫出有幾個步驟要做(有幾條link)

2. 考慮哪條link先做(找等號左邊的(被賦值的)，沒有出現在右邊(副職給別人))

```

```

```

```

Iterative 到 Recursive其實沒有這麼難！

iterative的方法理解之後，

轉成recursive其實只是把要更新的值傳進function裡，

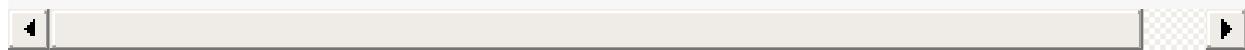
比如說這題，Iterative更新的方式是透過 py pre = pre.next.next

那要改成Recursive時只需要把這個步驟傳進Recursive function裡！

-> self.recurSwap(pre.next.next)，其他code都可以不變！！

Code

Recursive



```
py
class Solution(object):
    def swapPairs(self, head):
        dum = ListNode(-1)
        dum.next = head
        pre = dum
        #self.iterSwap(pre)
        self.recurSwap(pre)
        return dum.next
```

```
def recurSwap(self, pre):
    if not pre:
        return
    if pre.next and pre.next.next:
        first = pre.next
        second = pre.next.next

        pre.next = second
        first.next = second.next
        second.next = first
        self.recurSwap(pre.next.next)

def iterSwap(self, pre):
    while pre.next and pre.next.next:
        first = pre.next
```

```
second = pre.next.next

pre.next = second
first.next = second.next
second.next = first
pre = pre.next.next
```

Iterative

```
py
class Solution:
def swapPairs(self, head: ListNode) -> ListNode:
dum = ListNode(0)
dum.next = head
prev = dum
while prev.next and prev.next.next:
first = prev.next
second = prev.next.next
prev.next = second
first.next = second.next
second.next = first
```

```
    prev = prev.next.next
    return dum.next
```

```
## 30. Substring with Concatenation of All Words / 10/22
You are given a string, s, and a list of words, words, that are all of the same len

```

思路

由 i 挑選起點，一次比對一個單詞長度，每次移動一個單詞

計算總共需要的單詞長度 n

如果 j 迴圈執行完時，count 的長度 等於 n

表示這個區間的所有字都符合所需的字

Code

```

py
from collections import Counter
class Solution:
    def findSubstring(self, s: str, words: List[str]) -> List[int]:
        if not words: return []
        leng = len(words[0])
        n = len(words)
        words_dic = Counter(words)
        res = []
        print(len(s) - lengn) for i in range(len(s) - lengn + 1):
            str_map = dict()
            count = 0
            for j in range(n): # 字典裡有幾個單詞
                word = s[i+j:leng : i+leng + leng] # 一次比對一個單詞長度, 每次移動一個單詞
                if word not in words_dic:
                    break
                else:
                    if word not in str_map:
                        str_map[word] = 1
                    else:
                        str_map[word] += 1

```

```

                if str_map[word] > words_dic[word]:
                    break
                count += 1
                if count == n: # find all word
                    res.append(i)
            return res

```

--
25. Reverse Nodes in k-Group | 10/23 // TODO: re-understanding, 好難...

Given a linked list, reverse the nodes of a linked list k at a time and return its
k is a positive integer and is less than or equal to the length of the linked list.

Example:

Given this linked list: 1->2->3->4->5

For k = 2, you should return: 2->1->4->3->5

For k = 3, you should return: 3->2->1->4->5

Note:

Only constant extra memory is allowed.

You may not alter the values in the list's nodes, only nodes itself may be changed.
思路

- 如何寫Reverse LinkedList的Template:

- 找線索，最後一行通常是移動目標
 - 比如說這題 cur 為下一次循環的判斷
 - cur = last.next <- 永久成立
 - 那麼第一行的開頭就是last.next
 - 後面的推演就如同：
 - while z
 - temp = x (x為第三個節點)
 - x = y
 - y = z
 - z = temp

Code



py

```
class Solution:  
    def reverseKGroup(self, head: ListNode, k: int) -> ListNode:  
        if not head: return head  
        dum = ListNode(-1); dum.next = head  
        pre, cur = dum, head  
        i = 0  
        while cur:  
            i += 1  
            if i % k == 0:  
                pre = self.reverseOneGroup(pre, cur.next)  
                cur = pre.next  
            else:  
                cur = cur.next  
        return dum.next
```

```
def reverseOneGroup(self, pre, nxt):  
    last = pre.next  
    cur = last.next  
  
    while cur != nxt:  
        last.next = cur.next # cannot use temp here, cuz we have to maintain the st  
        cur.next = pre.next  
        pre.next = cur  
        cur = last.next
```

```
    return last
```

--
237. Delete Node in a Linked List / 10/23
Write a function to delete a node (**except** the tail) in a singly linked list, given

Given linked list -- head = [4,5,1,9], which looks like following:

```

```

思路

沒有給head，只給你刪除的節點叫你做操作

那就是把他的值覆蓋掉

並且也把他的next給覆蓋掉

Code

```
py
```

```
def deleteNode(self, node):  
    node.val = node.next.val  
    node.next = node.next.next
```

--
92. Reverse Linked List II / 10/27

Reverse a linked list from position m to n. Do it in one-pass.

Note: 1 ≤ m ≤ n ≤ length of list.

Example:

Input: 1->2->3->4->5->NULL, m = 2, n = 4

Output: 1->4->3->2->5->NULL

思路

reverse的寫法與上題一樣

但注意停止點應該要用second，

如果用first做停止點，second有機會是Null然後Null.next就爆了

```
### Code
```

```
py
class Solution:
    def reverseBetween(self, head: ListNode, m: int, n: int) -> ListNode:
        dum = ListNode(-1)
        dum.next = head
        pre = dum
        k = n-m
        while pre.next:
            m -= 1
            if m == 0:
                end = pre.next
            for _ in range(k+1): # Set the stop point to second
                end = end.next
            pre = self.reverse(pre,end)
            break
        else:
            pre = pre.next
        return dum.next
```

```
def reverse(self,pre,end):
    first = pre.next
    second = first.next

    while second != end:
        first.next = second.next
        second.next = pre.next
        pre.next = second
        second = first.next
    return second
```

```
---
## 339. Nested List Weight Sum | 10/27
Given a nested list of integers, return the sum of all integers in the list weight

Each element is either an integer, or a list -- whose elements may also be integers

Example 1:

Input: [[1,1],2,[1,1]]
Output: 10
```

```
Explanation: Four 1's at depth 2, one 2 at depth 1.
```

Example 2:

Input: [1,[4,[6]]]

Output: 27

```
Explanation: One 1 at depth 1, one 4 at depth 2, and one 6 at depth 3; 1 + 4*2 + 6*
```

技巧

- in Recursive Part:
 - 注意呼叫遞迴時的sum值要傳現值還是傳0
- in Iterative Part:
 - 兩個有關聯的東西可以使用Tuple
 - 此例: sublist 跟 Depth深度

思路

Recursive就是很Naive的思路

Iterative:

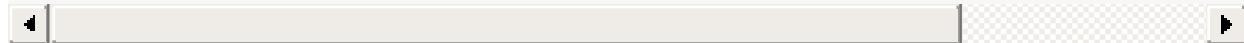
概念上就是使用tuple+stack

同一層的元素擁有同樣的深度

所以用Tupple把他們包在一起

Code

Recursive:



py

```
class Solution:  
    def depthSum(self, nestedList: List[NestedInteger]) -> int:  
        def dfs(sublist, depth, sum1):  
            for ei in sublist:  
                if ei.isInteger():  
                    sum1 += depth*ei.getInteger()  
                else:  
                    sum1 = dfs(ei.getList(), depth+1, sum1)  
            return sum1
```

```
sum1 = 0 # [[1,1],2,[1,1]]  
for ei in nestedList: #[1,1]  
    if ei.isInteger():  
        sum1 += ei.getInteger()
```

```
        else:
            sum1 += dfs(ei.getList(), 2, 0)
    return sum1
```

Iterative:

```
py
class Solution:
def depthSum(self, nestedList: List[NestedInteger]) -> int:
stack = []
res = 0
for ei in nestedList:
stack.append((ei, 1)) # first layer
```

```
while stack:
    top, depth = stack.pop()
    if top.isInteger():
        res += top.getInteger() * depth
    else:
        for ei in top.getList():
            stack.append((ei, depth+1))
return res
```

70. Climbing Stairs / 10/28

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 clim

Note: Given n will be a positive integer.

Example 1:

Input: 2

Output: 2

Explanation: There are two ways to climb to the top.

1. 1 step + 1 step

2. 2 steps

Example 2:

Input: 3

Output: 3

Explanation: There are three ways to climb to the top.

1. 1 step + 1 step + 1 step

2. 1 step + 2 steps

3. 2 steps + 1 step

技巧

- 導入python 內建的cache:

- `from functools import lru_cache`
- `@lru_cache(None)`

-

思路

1. **Top Down** Brute Force, Recursive

找規律，並記錄

`4 = G(3) + G(2)`

先知道了4，再從4開始往下拆解

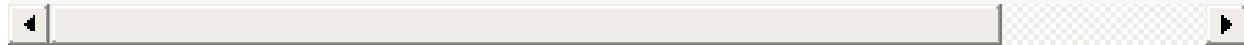
2. **Bottom Up** DP

`dp[i] = dp[i-1] + dp[i-2]`

不管n是多少，從底下往上建上去

Code

Brute-force Recursive:



py

class Solution:

def climbStairs(self, n: int) -> int:

def recur(n):

if str(n) in dic.keys():

return dic[str(n)]

one_step = recur(n-1)

dic[str(n-1)] = one_step

```

        two_step = recur(n-2)
        dic[str(n-2)] = two_step

    return one_step+two_step

dic = dict()
dic["1"] = 1
dic["2"] = 2
print(dic)
return recur(n)

```

```

```py
def dp_way(self,n):
 if n == 1:
 return 1
 dp = [0] * n
 dp[0], dp[1] = 1, 2

 for i in range(2, n):
 dp[i] = dp[i-1] + dp[i-2]

 return dp[n-1]

```

## 91. Decode Ways | 10/28

A message containing letters from A-Z is being encoded to numbers using the following mapping:

'A' -> 1

'B' -> 2

...

'Z' -> 26

Given a non-empty string containing only digits, determine the total number of ways to decode it.

Example 1:

Input: "12"

Output: 2

Explanation: It could be decoded as "AB" (1 2) or "L" (12).

Example 2:

Input: "226"

Output: 3

Explanation: It could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).

## 思路

這道題要求解碼方法，跟之前那道 Climbing Stairs 非常的相似，但是還有一些其他的限制條件，比如說一位數時不能為0，兩位數不能大於 26，其十位上的數也不能為0，除去這些限制條件，跟爬梯子基本沒啥區別

$dp[i]$  表示  $s$  中前  $i$  個字符組成的子串的解碼方法的個數，長度比輸入數組長多 1，並將  $dp[0]$  初始化為 1

所以 0 是個很特殊的存在，若當前位置是 0，那麼一定無法單獨拆分出來，即不能加上  $dp[i-1]$ ，就只能看否跟前一個數字組成大於等於 10 且小於等於 26 的數，能的話可以加上  $dp[i-2]$ ，否則就只能保持為 0 了

## Code

```
class Solution:
 def numDecodings(self, s: str) -> int:
 if s[0] == "0": return 0
 dp = [0] * (len(s)+1) # 因為是站在後面往前看，所以要多一個空間
 dp[0] = 1

 # 站在下一位往前看 所以才會 i-1，其實意思就是個位數
 for i in range(1, len(s)+1):
 if s[i-1] != "0": # 步伐一步，個位數為零不能單拆，但扔然有機會組成10 or 20
 dp[i] += dp[i-1]

 # 加上十位數之後 range 必須在10~26之間
 if i > 1 and "10" <= s[i-2:i] and s[i-2:i] <= "26":
 dp[i] += dp[i-2]

 return dp[-1]
```

## 202. Happy Number | 11/3

(Easy)

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:

**Input:** 19

**Output:** true

**Explanation:**

$$1^2 + 9^2 = 82$$

$$8^2 + 2^2 = 68$$

$$6^2 + 8^2 = 100$$

$$1^2 + 0^2 + 0^2 = 1$$

## 思路

## Code

```
class Solution:
 def isHappy(self, n: int) -> bool:
 def add(num):
 str_num = str(num)
 total = 0
 if str_num not in dic:
 dic[str_num] = 1

 for c in str_num:
 total += int(c)*int(c)

 if total == 1:
 return 0
 return total
 else:
 return -1
```

```
dic = dict()
status = n
while status > 0:
 status = add(status)

if status == 0:
 return True
else:
 return False
```

## 80. Remove Duplicates from Sorted Array II | 11/3

Given a sorted array `nums`, remove the duplicates in-place such that duplicates appeared at most twice and return the new length.

Do not allocate extra space for another array, you must do this by modifying the input array in-place with O(1) extra memory.

### Example 1:

```
Given nums = [1,1,1,2,2,3],

Your function should return length = 5, with the first five elements of
nums being 1, 1, 2, 2 and 3 respectively.

It doesn't matter what you leave beyond the returned length.
```

### Example 2:

```
Given nums = [0,0,1,1,1,1,2,3,3],

Your function should return length = 7, with the first seven elements of
nums being modified to 0, 0, 1, 1, 1, 2, 3 and 3 respectively.

It doesn't matter what values are set beyond the returned length.
```

### Clarification:

Confused why the returned value is an integer but your answer is an array?

Note that the input array is passed in by **reference**, which means modification to the input array will be known to the caller as well.

Internally you can think of this:

```
// nums is passed in by reference. (i.e., without making a copy)
int len = removeDuplicates(nums);

// any modification to nums in your function would be known by the caller.
// using the length returned by your function, it prints the first len
elements.
for (int i = 0; i < len; i++) {
 print(nums[i]);
}
```

## 思路

把何時狀態該如何轉換要想清楚再開始作答！

## Code

```
.....
[1,1,2,2,2,3,3]

[1,1,2,2,3,3,3]

.....
class Solution:
 def removeDuplicates(self, nums):
 if (len(nums) < 2):

 return len(nums)
 cur, nex = 0, 1
 counter = 0
 while nex != len(nums):
 if nums[cur] == nums[nex]:
 if counter == 0:
 counter += 1
 nums[cur+1] = nums[nex]
 cur += 1
 else:
 nums[cur + 1] = nums[nex]
 cur += 1
```

```
 counter = 0
 nex += 1

 return cur+1
```

## 394. Decode String | 11/4

Given an encoded string, return its decoded string.

The encoding rule is: k[encoded\_string], where the encoded\_string inside the square brackets is being repeated exactly k times. Note that k is guaranteed to be a positive integer.

You may assume that the input string is always valid; No extra white spaces, square brackets are well-formed, etc.

Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers, k. For example, there won't be input like 3a or 2[4].

Examples:

s = "3[a]2[bc]", return "aaabcabc".

s = "3[a2[c]]", return "accaccacc".

s = "2[abc]3[cd]ef", return "abcabcccdcdcdef".

## 思路

DEBUG THIS

## Code

```
class Solution:
 def decodeString(self, s: str) -> str:
 int_stack, char_stack = [], []
 int_hold, char_hold = 0, ""
 reading_char = False

 for c in s:
 print(char_stack)
 if c == "[":
 int_stack.append(int_hold)
 char_stack.append(char_hold)
 int_hold = 0
 char_hold = ""
```

```

 int_stack.append(int_hold)
 int_hold = 0
 if reading_char:
 char_stack.append(char_hold)
 char_hold = ""
 reading_char = False
 else:
 reading_char = True

 elif c == "]":

 top_int = int_stack.pop()

 temp_str = char_hold * top_int
 if char_stack:
 top_char = char_stack.pop()
 temp_str = top_char + temp_str
 char_hold = temp_str # TODO: DEBUG
 char_stack.append(temp_str)
 reading_char = False

 elif c.isdigit():
 int_hold = int_hold*10 + int(c)
 else:
 char_hold += c

 if len(int_stack) == 0 and len(char_stack) == 1:
 return char_stack.pop()
 else:
 return "WA"

```

---

## Mathwork.Custom Sorted Array | 11/12

---

Given an unsorted array,

Output an array which let all even number before odd number

And return the min move

<https://www.geeksforgeeks.org/segregate-even-and-odd-numbers/>

思路

Code

....  
Given an unsorted array,  
Output an array which let all even number before odd number  
And return the min move

<https://www.geeksforgeeks.org/segregate-even-and-odd-numbers/>

....

```
def even_before_odd(arr):
 print("original",arr)
 l, r = 0, len(arr)-1
 count = 0

 while l < r:

 while arr[l] % 2 == 0 and l < r:
 l += 1

 while arr[r] % 2 != 0 and l < r:
 r -= 1

 if r != l:
 count += 1
 arr[l], arr[r] = arr[r], arr[l]

 print(arr)
 print(count)
 print("___")

if __name__ == '__main__':
 even_before_odd([1,2,3]) # 1
 even_before_odd([1,2,3,4]) # 1
 even_before_odd([1,3,5]) # 0
 even_before_odd([1,3]) # 0
 even_before_odd([1]) # 0
 even_before_odd([2,4]) # 0
 even_before_odd([2,4,1,3]) # 0
 even_before_odd([]) # 0
 even_before_odd([2,4,1,3,6,8,10]) # 2
```

---

## 1130. Minimum Cost Tree From Leaf Values | 11/13

---

Given an array arr of positive integers, consider all binary trees such that:

Each node has either 0 or 2 children;

The values of arr correspond to the values of each leaf in an in-order traversal of the tree.  
(Recall that a node is a leaf if and only if it has 0 children.)

The value of each non-leaf node is equal to the product of the largest leaf value in its left and right subtree respectively.

Among all possible binary trees considered, return the smallest possible sum of the values of each non-leaf node. It is guaranteed this sum fits into a 32-bit integer.

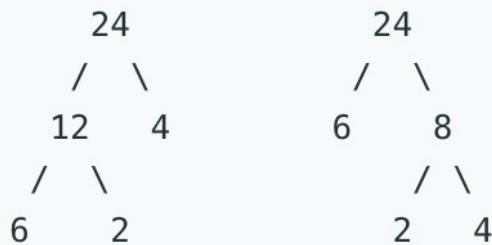
### Example 1:

**Input:** arr = [6,2,4]

**Output:** 32

#### Explanation:

There are two possible trees. The first has non-leaf node sum 36, and the second has non-leaf node sum 32.



### Constraints:

- $2 \leq \text{arr.length} \leq 40$
- $1 \leq \text{arr}[i] \leq 15$
- It is guaranteed that the answer fits into a 32-bit signed integer (ie. it is less than  $2^{31}$ ).

## 技巧

透過index去操作list，

如果牽涉到移除該陣列元素時會很好用！

取得陣列中最小元素 : arr.index(min(arr))

操作完後pop掉 : arr.pop(idx)

## 思路

最小兩數相乘會有最小結果

## Code

```
class Solution:
 def mctFromLeafValues(self, arr: List[int]) -> int:
 res = 0
 while len(arr) > 1:
 min_idx = arr.index(min(arr))
 if 0 < min_idx < len(arr)-1:
 res += min(arr[min_idx-1], arr[min_idx+1]) * arr[min_idx]
 else:
 res += arr[1] if min_idx == 0 else arr[min_idx-1] * arr[min_idx]
 arr.pop(min_idx)
 return res
```

## Mathwork. Max Value Among Shortest Distance in a Matrix | 11/13

Given a grid with w as width, h as height.

Each cell of the grid represents a potential building lot and we will be adding "n" buildings inside this grid.

The goal is for the furthest of all lots to be as near as possible to a building.

Given an input n, which is the number of buildings to be placed in the lot,  
determine the building placement to minimize the distance of the most distant empty lot is from the building.

For example, `w=4, h=4` and `n=3`. An optimal grid placement sets any lot within two unit distance of the building. The answer for this case is 2.

"0" indicates optimal building placement and in this case the maximal value of all shortest distances to the closest building for each cell is "2".

```
1 0 1 2
2 1 2 1
1 0 1 0
2 1 2 1
```

The above represents one optimal solution, there could be more like the above array rotated as an example. The above is an optimal solution because out of the 3 buildings ( $n=3$ ), one building was placed at index  $(0,1)$ , second was placed at  $(2,1)$  and third was placed at  $(2,3)$ . The surrounding horizontal and vertical distance is shown as 1 and 2 by adding 1 each time we move horizontally and/or vertically. Note again that diagonal movement is not allowed:

```
1 ← 0 → 1 → 2
↓
2 ← 1 → 2 ← 1
↑ ↑
1 ← 0 → 1 ← 0
↓ ↓
2 ← 1 → 2 ← 1
```

## 思路

## Code

# 300. Longest Increasing Subsequence | 11/14

Given an unsorted array of integers, find the length of longest increasing subsequence.

Example:

Input: [10,9,2,5,3,7,101,18]

Output: 4

Explanation: The longest increasing subsequence is [2,3,7,101], therefore the length is 4.

Note:

There may be more than one LIS combination, it is only necessary for you to return the length.  
Your algorithm should run in  $O(n^2)$  complexity.

Follow up: Could you improve it to  $O(n \log n)$  time complexity?

## 思路

新的結果會需要依賴舊的, 用dp array把previous status 紀錄起來

[10, 9, 2, 5, 3, 7, 101, 18].

dp: 0, 1, 2, 3, 4, 5, 6, 7

1, 1, 2, 2, 3, 4, 4.

dp[5]

( $dp[2], dp[3], dp[4]$ ).  
 $dp[3] = \underline{dp[4]} > dp[2]$ .)

$dp[5] = dp[3] \text{ or } dp[4] + 1$ .

新的結果需要依賴舊的.

用dp array 把 previous status 紀錄起來.

## Code

```
def lengthOfLIS(self, nums: List[int]) -> int:
 dp = [1] * len(nums)
 res = 0
 for i in range(len(nums)):
 for j in range(0, i):
 if nums[j] < nums[i]:
 dp[i] = max(dp[i], dp[j] + 1)

 res = max(res, dp[i])

 return res
```

## Trie

### 208. Implement Trie (Prefix Tree) | 11/17

Implement a trie with insert, search, and startsWith methods.

Example:

```
Trie trie = new Trie();

trie.insert("apple");

trie.search("apple"); // returns true

trie.search("app"); // returns false

trie.startsWith("app"); // returns true

trie.insert("app");

trie.search("app"); // returns true
```

Note:

You may assume that all inputs are consist of lowercase letters a-z.

All inputs are guaranteed to be non-empty strings.

## 思路

Trie 實作:

- 根節點為空
- 每個子結點都包含一個字典裝 (char: TrieNode)
- 每個子結點也包含一個isWord(), 來顯示是否已經形成一個單字

## Code

```
class Trie:

 def __init__(self):
 """
 Initialize your data structure here.
 """
 self.root = TrieNode()

 def insert(self, word: str) -> None:
 """
 Inserts a word into the trie.
 """
 cur = self.root
 for c in word:
 if c not in cur.dic.keys():
 cur.dic[c] = TrieNode()
 cur = cur.dic[c]
 cur.isWord = True

 def search(self, word: str) -> bool:
 """
 Returns if the word is in the trie.
 """
 cur = self.root
 for c in word:
 if c not in cur.dic.keys():
 return False
 cur = cur.dic[c]
 if cur.isWord:
 return True
 else:
 return False

 def startsWith(self, prefix: str) -> bool:
 """

```

```

 Returns if there is any word in the trie that starts with the given prefix.
 """
 cur = self.root
 for c in prefix:
 if c not in cur.dic.keys():
 return False
 cur = cur.dic[c]
 return True

class TrieNode():

 def __init__(self):
 self.dic = dict()
 self.isWord = False
 """

Your Trie object will be instantiated and called as such:
obj = Trie()
obj.insert(word)
param_2 = obj.search(word)
param_3 = obj.startsWith(prefix)
"""

```



## 212. Word Search II | 11/18

---

### 技巧

- 在board裏搜尋出所有字串, 使用next()來取值:

```

m, n = len(board), len(board[0])
for i in range(m):
 #print(board[i].index("a"))
 a = (j for j, n in enumerate(board[i]) if n == "a")
 while a:
 print(next(a)) #

```

### 思路

如果在board中搜尋字串複雜度太高

使用Trie + 反向思考來解

遍歷board裡的所有char 及其鄰居(dfs)

來試圖在Trie中匹配word

使用visited來記錄訪問過的位置及其代表的文字

當搜尋到isEnd()時把走過的字join起來後放進return set()中

## Code

```
class Trie:
 def __init__(self):
 self.root = TrieNode()

 def insert(self, word):
 cur = self.root

 for c in word:
 if c not in cur.children:
 cur.children[c] = TrieNode()
 cur = cur.children[c]
 cur.isEnd = True

class TrieNode:
 def __init__(self):
 self.children = dict()
 self.isEnd = False

class Solution:
 def findWords(self, board: List[List[str]], words: List[str]) -> List[str]:
 trie = Trie()
 for word in words:
 trie.insert(word)

 found = set()
 for r in range(len(board)):
 for c in range(len(board[0])):
 self.traverse(trie.root, r, c, found, dict(), board)
 return found

 def traverse(self, trie_node, r, c, found, visited, board):
 if trie_node.isEnd:
 found.add(''.join(visited.values()))

 if r < 0 or r >= len(board) or c < 0 or c >= len(board[0]) or (board[r][c] not in trie_node.children):
 return

 visited[(r,c)] = board[r][c]
 self.traverse(trie_node.children[board[r][c]], r+1, c, found, visited, board)
 self.traverse(trie_node.children[board[r][c]], r-1, c, found, visited, board)
 self.traverse(trie_node.children[board[r][c]], r, c+1, found, visited, board)
 self.traverse(trie_node.children[board[r][c]], r, c-1, found, visited, board)
```

```
del visited[(r,c)]
```

## 16. 3Sum Closest | 11/20

Given an array nums of n integers and an integer target, find three integers in nums such that the sum is closest to target. Return the sum of the three integers. You may assume that each input would have exactly one solution.

Example:

Given array nums = [-1, 2, 1, -4], and target = 1.

The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

### 思路

跟3Sum一樣的解法

但這題不一定有正確的答案, 只求最接近的

因此需要另一個diff來幫助我們更新當前的closest

還需要一個if來判斷當下該移動left還是移動right

條件就是“下一個可能解會怎麼出現”

### Code

```
class Solution:
 def threeSumClosest(self, nums: List[int], target: int) -> int:
 nums = sorted(nums)
 closest = nums[0] + nums[1] + nums[2]
 diff = abs(target - closest)

 for i in range(len(nums) - 2):
 l, r = i + 1, len(nums)-1

 while l < r:
 new_sum = nums[i] + nums[l] + nums[r]
 new_diff = abs(target - new_sum)
 if new_diff < diff:
```

```
 closest, diff = new_sum, new_diff

 if new_sum < target:
 l += 1
 else:
 r -= 1
 return closest
```

## 18. 4Sum | 11/20

Given an array `nums` of  $n$  integers and an integer `target`, are there elements  $a$ ,  $b$ ,  $c$ , and  $d$  in `nums` such that  $a + b + c + d = \text{target}$ ? Find all unique quadruplets in the array which gives the sum of `target`.

Note:

The solution set must not contain duplicate quadruplets.

Example:

```
Given array nums = [1, 0, -1, 0, -2, 2], and target = 0.
```

```
A solution set is:
```

```
[
 [-1, 0, 0, 1],
 [-2, -1, 1, 2],
 [-2, 0, 0, 2]
]
```

## 思路

基本上就是3Sum的延伸, 多一層for而已

難度在於該如何跳過重複

排序後要是起點一樣, 搜尋結果就會是一樣的, 因此可以在兩處加上檢查

檢查的概念是, 在第二次循環時, 起點不能跟上一次一樣

1. 在for(i)底下
  - 簡單用*i>0*就可以
2. 在for(j)底下
  - 不能使用*j>0!!* 在[0,0,0,0]這個測資會fail
  - 要使用*j>i+1*, 這個概念才是j的第二次循環

然而這兩個檢查點還不夠,

在while循環l跟r時還是可能會出現此l/r跟上個l/r是一樣的

因此在 total == target 條件底下還要放個跳過重複的循環

Tricky task case:

[−1,0,−5,−2,−2,−4,0,1,−2], t = −9

[0,0,0,0], t = 0

## Code

```
class Solution:
 def fourSum(self, nums: List[int], target: int) -> List[List[int]]:
 res = []
 nums = sorted(nums)
 for i in range(len(nums)-3):
 if i > 0 and nums[i] == nums[i-1]: continue
 for j in range(i+1, len(nums)-2):
 if j > i+1 and nums[j] == nums[j-1]: continue

 l, r = j+1, len(nums)-1

 while l < r:
 total = nums[l] + nums[r] + nums[i] + nums[j]
 if total == target:
 comb = [nums[l]] + [nums[r]] + [nums[i]] + [nums[j]]
 res.append(comb)
 while l < r and nums[l] == nums[l+1]: l+=1 # TODO: Tri
 while l < r and nums[r] == nums[r-1]: r-=1
 l += 1
 r -= 1
 elif total < target:
 l += 1
 else:
 r -= 1

 return res
```

## N-Sum Extendable Solution

```
def fourSum(self, nums, target):
 def findNsum(nums, target, N, result, results):
 if len(nums) < N or N < 2 or target < nums[0]*N or target > nums[-1]*N: #
 return
 if N == 2: # two pointers solve sorted 2-sum problem
 l,r = 0,len(nums)-1
 while l < r:
 s = nums[l] + nums[r]
 if s == target:
 results.append(result + [nums[l], nums[r]])
 l += 1
 while l < r and nums[l] == nums[l-1]:
 l += 1
 elif s < target:
 l += 1
 else:
 r -= 1
 else: # recursively reduce N
 for i in range(len(nums)-N+1):
 if i == 0 or (i > 0 and nums[i-1] != nums[i]):
 findNsum(nums[i+1:], target-nums[i], N-1, result+[nums[i]], res
results = []
findNsum(sorted(nums), target, 4, [], results)
return results
```

## 31. Next Permutation | 11/21

Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers.

If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

The replacement must be in-place and use only constant extra memory.

Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column.

1,2,3 → 1,3,2

3,2,1 → 1,2,3

1,1,5 → 1,5,1

## 思路

這道題讓我們求下一個排列順序，由題目中給的例子可以看出來，如果給定數組是降序，則說明是全排列的最後一種情況，則下一個排列就是最初始情況，可以參見之前的博客 Permutations。再來看下面一個例子，有如下的一個數組

1    2    7    4    3    1

下一個排列為：

1    3    1    2    4    7

那麼是如何得到的呢，我們通過觀察原數組可以發現，如果從末尾往前看，數字逐漸變大，到了2時才減小的，然後再從後往前找第一個比2大的數字，是3，那麼我們交換2和3，再把此時3後面的所有數字轉置一下即可，步驟如下：

1    2    7    4    3    1

1    2    7    4    3    1

1    3    7    4    2    1

1    3    1    2    4    7

## Code

```
class Solution:
 def nextPermutation(self, nums: List[int]) -> None:
 """
 Do not return anything, modify nums in-place instead.
 """

 for i in range(len(nums)-1, -1, -1):
 if i == 0:
 mid = len(nums) // 2
 for j in range(mid):
 nums[j], nums[-(j+1)] = nums[-(j+1)], nums[j]
 if i >= 0 and nums[i] > nums[i-1]:
 # Step1. Swapping
 for j in range(len(nums)-1, i-1, -1):
 if nums[j] > nums[i-1]:
 nums[i-1], nums[j] = nums[j], nums[i-1]
 break
 # Step2. Reordering after swapping
```

```
swap_times = (len(nums) - i +1) //2
for j in range(swap_times):
 nums[i+j], nums[-(j+1)] = nums[-(j+1)], nums[i+j]

Step3. Break
break
```

## 32. Longest Valid Parentheses | 11/21

Given a string containing just the characters '(' and ')', find the length of the longest valid (well-formed) parentheses substring.

Example 1:

Input: "()"

Output: 2

Explanation: The longest valid parentheses substring is "()"

Example 2:

Input: ")()())"

Output: 4

Explanation: The longest valid parentheses substring is "())"

### 思路

是要求"連續合法的括號組合的最長"

反過來思考: 何時被中斷

1. 多餘的左瓜
  - 沒有右瓜可以配對, 在stack中剩下了
2. 多餘的右瓜
  - 沒有左瓜可以配對

那麼標記出多餘的位置就很重要, 因此stack裡放左瓜的index

標記完之後遍歷index\_list, 求出最大的連續合法

## Code

```
class Solution:
 def longestValidParentheses(self, s: str) -> int:
 stack = []
 idx_list = [0] * len(s)
 for i,p in enumerate(s):
 if p == '(':
 stack.append(i)
 elif p == ')':
 if stack:
 valid_idx = stack.pop() # matching w/ '('
 idx_list[valid_idx] = 1 # mark that pos w/ 1
 else:
 idx_list[i] = -1 # mark as invalid, redundant ')'

 # mark as invalid, redundant '('
 while stack:
 invalid_idx = stack.pop()
 idx_list[invalid_idx] = -1

 counter_list = [0] # initialize inorder to call max()
 counter = 0
 for idx in idx_list:
 if idx < 0:
 counter_list.append(counter) # append current max
 counter = 0 # reset the continuous count
 else:
 if idx == 1:
 counter += 1

 counter_list.append(counter) # append the rest val in counter
 return max(counter_list)*2
```

## 36. Valid Sudoku(合法數獨) | 11/21

Determine if a 9x9 Sudoku board is valid. Only the filled cells need to be validated according to the following rules:

Each row must contain the digits 1-9 without repetition.

Each column must contain the digits 1-9 without repetition.

Each of the 9 3x3 sub-boxes of the grid must contain the digits 1-9 without repetition.

5	3	.	.	7	.	.	.	.
6	.	.	1	9	5	.	.	.
.	9	8	.	.	.	6	.	.
8	.	.	6	.	.	.	.	3
4	.	8	.	3	.	.	.	1
7	.	.	2	.	.	.	.	6
.	6	.	.	.	2	8	.	.
.	.	4	1	9	.	.	.	5
.	.	.	8	.	.	7	9	.

A partially filled sudoku which is valid.

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

### Example 1:

#### Input:

```
[
 ["5","3",".",".","7",".",".",".","."],
 ["6",".",".","1","9","5",".",".","."],
 [".","9","8",".",".",".","6","."],
 ["8",".",".",".","6",".",".",".","3"],
 ["4",".",".","8",".","3",".",".","1"],
 ["7",".",".",".","2",".",".",".","6"],
 [".","6",".",".",".","2","8","."],
 [".",".",".","4","1","9",".",".","5"],
 [".",".",".","8",".",".","7","9"]
]
```

## Example 2:

### Input:

```
[
 ["8","3",".",".","7",".",".",".","."],
 ["6",".",".","1","9","5",".",".","."],
 [".","9","8",".",".",".","6","."],
 ["8",".",".","6",".",".",".","3"],
 ["4",".",".","8",".","3",".",".","1"],
 ["7",".",".","2",".",".",".","6"],
 [".","6",".",".",".","2","8","."],
 [".",".","4","1","9",".",".","5"],
 [".",".","8",".","7","9"]
]
```

### Output:

**Explanation:** Same as Example 1, except with the 5 in the top left corner being

modified to 8. Since there are two 8's in the top left 3x3 sub-box, it is invalid.

### Note:

- A Sudoku board (partially filled) could be valid but is not necessarily solvable.
- Only the filled cells need to be validated according to the mentioned rules.
- The given board contain only digits 1-9 and the character '.'.
- The given board size is always 9x9.

## 思路

## Code

```
class Solution:
 def isValidSudoku(self, board: List[List[str]]) -> bool:
 if len(board) != 9 or len(board[0]) != 9: return False
```

```

Sub-box check
Define Center
center = [(1,1), (1,4), (1,7), (4,1), (4,4), (4,7), (7,1), (7,4), (7,7)]
Define Location
direction = [(0,0), (0,1), (1,0), (0,-1), (-1,0), (1,1), (-1,-1), (1,-1), (
Check all Center with direction
for c in center:
 visited = []
 for d in direction:
 x = c[0] + d[0]
 y = c[1] + d[1]
 if board[x][y] == ".":
 continue
 if board[x][y] in visited:
 return False
 else:
 visited.append(board[x][y])
Row check
for r in range(9):
 visited = []
 for c in range(9):
 if board[r][c] == ".":
 continue
 if board[r][c] in visited:
 return False
 else:
 visited.append(board[r][c])
Column check
for c in range(9):
 visited = []
 for r in range(9):
 if board[r][c] == ".":
 continue
 if board[r][c] in visited:
 return False
 else:
 visited.append(board[r][c])
return True

```

## 37. Sudoku Solver | 11/21

解一個數獨

保證有一解

思路

TODO: Debug The "ref" in isValid()

iter 時也要return, 這樣才能把所有狀態串在一起

把每個空格帶入數字, 並且視為一個狀態

在遞迴這個狀態之前先去檢查是否Valid, 不Valid直接帶入下個數字

成功走到終點的狀態會全部return True

如果這個狀態在未來的某條路斷了,

要把這個狀態還原成".",

這樣其他狀態的人才能判斷他是".", 並修改他

isValid() 也是一個厲害的寫法 記得看看

## Code

```
class Solution:
 def solveSudoku(self, board: List[List[str]]) -> None:
 """
 Do not return anything, modify board in-place instead.
 """
 self.recur(0, 0, board)

 # iter through column order(left2right, up2down)
 def recur(self, r, c, board):
 # Meet Line10, no longer need to validate
 if r == 9:
 return True
 # Meet row end, next line iter
 if c >= 9:
 return self.recur(r+1, 0, board)
 # Not ".", keep iter
 if board[r][c] != ".":
 return self.recur(r, c+1, board)

 # Is ".", Assign 1~9, validate, then iter
 for i in range(1, 10):
 if not self.isValid(r, c, i, board): continue
 board[r][c] = str(i)

 if self.recur(r, c+1, board):
 return True
 # If invalid clean-up and Assign another num
```

```

 board[r][c] = "."
 return False

def isValid(self, r, c, val, board):
 val = str(val)
 # row
 for i in range(9):
 if board[r][i] == val:
 return False

 # column
 for i in range(9):
 if board[i][c] == val:
 return False
 # 3*3 box
 # Use left corner as reference point
 # Decide where is the section by mod

 #ref = (r%3, c%3)
 row = r - r % 3
 col = c - c % 3
 for i in range(3):
 for j in range(3):
 #if board[i+3*ref[0]][j+3*ref[1]] == val:
 if board[i+row][j+col] == val:
 return False
 return True

```

## 43. Multiply Strings | 11/23

---

Given two non-negative integers num1 and num2 represented as strings, return the product of num1 and num2, also represented as a string.

Example 1:

Input: num1 = "2", num2 = "3"

Output: "6"

Example 2:

Input: num1 = "123", num2 = "456"

Output: "56088"

Note:

The length of both num1 and num2 is < 110.

Both num1 and num2 contain only digits 0-9.

Both num1 and num2 do not contain any leading zero, except the number 0 itself.

You must not use any built-in BigInteger library or convert the inputs to integer directly.

## 技巧

- map(func, list):
  - 功能: 將list(or other iterable)的元素帶入func運算
  - 回傳: 回傳一個map object, 需用一個container去承接
  - 應用: output\_list = list(map(func, input\_list))
  - 應用2: 將list中所有的int轉為str list
    - product = list(map(str, product[ptr:]))

## 思路

這種運算方式可以用於高位數的運算, 也不會overflow!!!

性質: num1\*num2 的位數不會超過len(num1)+len(num2)

如註釋所示 把本來一行的sub\_mult 拆成len(num2)行

然後再將pos\_num1向左位移一位

簡潔漂亮! 牛逼!

## Code

Elegant Solution

```
class Solution:
 def multiply(self, num1: str, num2: str) -> str:
 ...
 num2: 1 2 3
 num1: x 6 7 8

 2 4
 1 6
 8
 (n1_pos--)
```

```

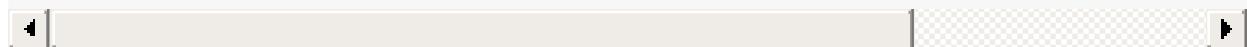
 2 1
 1 4
 7
.....
product = [0] * (len(num1)+len(num2))
n1_pos = len(product) - 1

for n1 in num1[::-1]:
 n2_pos = n1_pos
 for n2 in num2[::-1]:
 product[n2_pos] += int(n1) * int(n2) # ability to add the carry bit
 product[n2_pos - 1] += product[n2_pos]//10 # carry bit
 product[n2_pos] %= 10 # digit
 n2_pos -= 1 # move from 3 -> 2
 n1_pos -=1

remove leading 0
ptr = 0
while ptr < len(product)-1 and product[ptr] == 0: # keep at least 1 zero fo
 ptr += 1

product = list(map(str, product[ptr:])) # map() return a map object, need a
return "".join(product)

```



## Naive Elementary school math product

```

class Solution:
 def multiply(self, num1: str, num2: str) -> str:
 if num1 == "0" or num2 == "0":
 return "0"
 sub_mult_list = list()
 num1 = list(num1)[::-1]
 num2 = list(num2)[::-1]
 for i, d1 in enumerate(num1):
 sub_mult = []
 carry = 0
 for d2 in num2:
 mul = int(d2) * int(d1) + carry
 digit = mul%10
 carry = mul//10
 sub_mult.insert(0, str(digit))
 if carry != 0:
 sub_mult.insert(0, str(carry))

 # append 0
 for _ in range(i):
 sub_mult.append("0")
 sub_mult_list.append(sub_mult[:])

 # make every sub_mult in sub_mult_list have same digit
 max_lenth = len(sub_mult_list[-1])

```



```

for sub_mult in sub_mult_list:
 while len(sub_mult) != max_lenth:
 sub_mult.insert(0,"0")

add up all of the submult in submultlist
ret = []
carry = 0
for k in range(max_lenth-1, -1, -1):
 add = 0
 add += carry
 for i in range(len(sub_mult_list)):
 add += int(sub_mult_list[i][k])
 digit = add%10
 ret.insert(0,str(digit))

 carry = add//10

if carry != 0:
 ret.insert(0,str(carry))

return "".join(ret)

```

---

## 48. Rotate Image | 11/23

---

You are given an  $n \times n$  2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

Note:

You have to rotate the image in-place, which means you have to modify the input 2D matrix directly. DO NOT allocate another 2D matrix and do the rotation.

Example 1:

```
Given input matrix =
[
 [1,2,3],
 [4,5,6],
 [7,8,9]
,
```

```
rotate the input matrix in-place such that it becomes:
[
 [7,4,1],
 [8,5,2],
 [9,6,3]
,
```

### Example 2:

```
Given input matrix =
[
 [5, 1, 9,11],
 [2, 4, 8,10],
 [13, 3, 6, 7],
 [15,14,12,16]
,
```

```
rotate the input matrix in-place such that it becomes:
[
 [15,13, 2, 5],
 [14, 3, 4, 1],
 [12, 6, 8, 9],
 [16, 7,10,11]
```

1

思路

$i=0$	$i=1$
1 2 3	1 4 7
4 5 6	2 5 6
7 8 9	3 8 9

1 4 7	1 4 7	74 1
2 5 6	2 5 8	→ 85 2
3 8 9	3 6 9	96 3.

先將矩陣逆時針轉

再180度翻轉

## Code

```
class Solution:
 def rotate(self, matrix: List[List[int]]) -> None:
 """
 Step1: Step2:
 1 4 7 7 4 1
 2 5 8 8 5 2
 3 6 9 9 6 3
 """
 for i in range(len(matrix)):
 for j in range(i+1, len(matrix[0])):
 matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]

 for i in range(len(matrix)):
 matrix[i] = matrix[i][::-1]
```

可將下面的for與上面的合併，因為搞完第一層之後第一層的元素在下面不會用到了  
因此可以直接翻轉

```
class Solution:
 def rotate(self, matrix: List[List[int]]) -> None:
 """
 i=0:
 7 4 1
 2 5 8
 3 6 9
 """
 for i in range(len(matrix)):
 for j in range(i+1, len(matrix[0])):
```

```
 matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]
matrix[i] = matrix[i][::-1]
```

## 59. Spiral Matrix II | 11/24

Given a positive integer n, generate a square matrix filled with elements from 1 to n<sup>2</sup> in spiral order.

Example:

Input: 3

Output:

[

[ 1, 2, 3 ],

[ 8, 9, 4 ],

[ 7, 6, 5 ]

]

思路

Code

```
class Solution:
 def generateMatrix(self, n: int) -> List[List[int]]:
 ret = [[0] * n for _ in range(n)]

 left, up, right, down, num = 0, 0, n-1, n-1, 1

 while left <= right and up <= down:

 for i in range(left, right+1):
 ret[up][i] = num
 num += 1
 up += 1

 for i in range(up, down+1):
```

```

 ret[i][right] = num
 num += 1
 right -= 1

 for i in range(right, left-1, -1):
 ret[down][i] = num
 num += 1
 down -= 1

 for i in range(down, up-1, -1):
 ret[i][left] = num
 num += 1
 left += 1

 return ret

```

## 61. Rotate List | 11/24

---

Given a linked list, rotate the list to the right by k places, where k is non-negative.

Example 1:

Input: 1->2->3->4->5->NULL, k = 2

Output: 4->5->1->2->3->NULL

Explanation:

rotate 1 steps to the right: 5->1->2->3->4->NULL

rotate 2 steps to the right: 4->5->1->2->3->NULL

Example 2:

Input: 0->1->2->NULL, k = 4

Output: 2->0->1->NULL

Explanation:

rotate 1 steps to the right: 2->0->1->NULL

rotate 2 steps to the right: 1->2->0->NULL

rotate 3 steps to the right: 0->1->2->NULL

rotate 4 steps to the right: 2->0->1->NULL

## 思路

暴力法肯定超時,

又發現會有規律, 所以一定用得到mod,

用mod就得找這個list的長度,

這裡使用一個trick把頭尾接起來.

剩下的就是要找該在何處斷開.

由實驗反推推倒式:

1->2->3->4->5

time 為該斷開的點

k = 2, time = 2

k = 1, time = 3

我們要在前一個點斷開,

所以時機點為( length - k % length ) - 1

## Code

```
class Solution:
 def rotateRight(self, head: ListNode, k: int) -> ListNode:
 if k == 0 or not head:
 return head

 length = self.get_list_length(head)
 time = length - k % (length) - 1

 self.connect_tail2head(head)

 for i in range(time):
 head = head.next
 newHead = head.next
```

```
head.next = None

return newHead

def connect_tail2head(self, head):
 dum = head
 while head.next:
 head = head.next
 head.next = dum

def get_list_length(self, head):
 count = 0
 while head:
 count += 1
 head = head.next
 return count
```

---

## 62. Unique Paths | 11/25

---

A robot is located at the top-left corner of a  $m \times n$  grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

How many possible unique paths are there?



Above is a  $7 \times 3$  grid. How many possible unique paths are there?

**Note:**  $m$  and  $n$  will be at most 100.

### Example 1:

**Input:**  $m = 3, n = 2$

**Output:** 3

**Explanation:**

From the top-left corner, there are a total of 3 ways to reach the bottom-right corner:

1. Right  $\rightarrow$  Right  $\rightarrow$  Down
2. Right  $\rightarrow$  Down  $\rightarrow$  Right
3. Down  $\rightarrow$  Right  $\rightarrow$  Right

### Example 2:

**Input:**  $m = 7, n = 3$

**Output:** 28

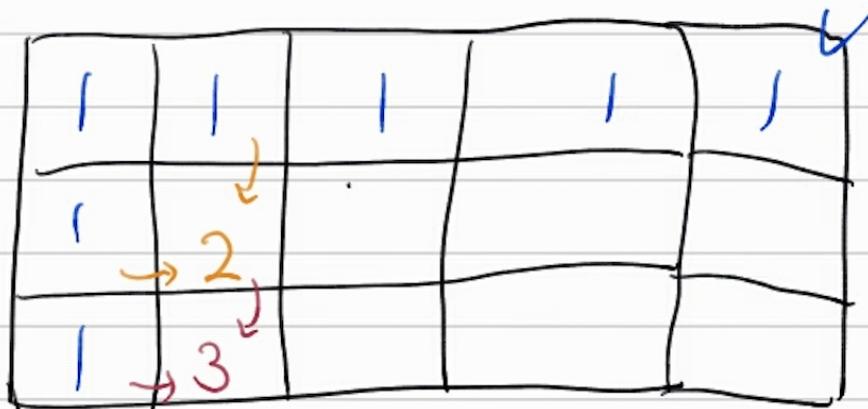
## 思路

非常經典的題型

反覆思考

現態、次態

# ⊗⊗⊗⊗⊗ Unique Path



：只有一种方法可以到達，so "1"

：從(0,1) or (1,0) 可達.  $1+1=2$

：從(1,1) or (2,0) 可達.  $2+1=3$ .

$$\Rightarrow dp[i][j] = dp[i-1][j] + dp[i][j-1],$$

Code

```
class Solution:
 def uniquePaths(self, m: int, n: int) -> int:
 dp = [[1] * n]*m

 for i in range(1, m):
 for j in range(1, n):
```

```
dp[i][j] = dp[i][j-1] + dp[i-1][j]
return dp[-1][-1]
```

---

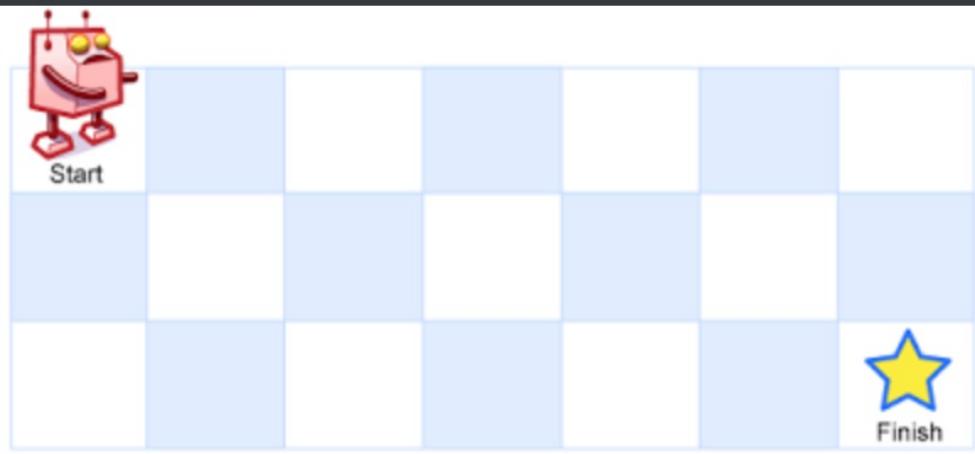
## 63. Unique Paths II | 11/25

---

A robot is located at the top-left corner of a  $m \times n$  grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

Now consider if some obstacles are added to the grids. How many unique paths would there be?



An obstacle and empty space is marked as `1` and `0` respectively in the grid.

**Note:**  $m$  and  $n$  will be at most 100.

**Example 1:**

**Input:**

```
[
 [0,0,0],
 [0,1,0],
 [0,0,0]
]
```

**Output:** 2

**Explanation:**

There is one obstacle in the middle of the  $3 \times 3$  grid above.

There are two ways to reach the bottom-right corner:

1. Right  $\rightarrow$  Right  $\rightarrow$  Down  $\rightarrow$  Down
2. Down  $\rightarrow$  Down  $\rightarrow$  Right  $\rightarrow$  Right

思路

0	0	0	0	0	0	0	0
0	1						
0							
0							

由於我們已經沒有辦法保證直走右或直走下，一定會只有一種解法(有可能出現障礙)

因此所有陣列數值只能初始化為0

意思就是沒有辦法初始化邊界，(邊界在這題也必須重新計算)

因此我們必須將起點往右下移動一個位置即(1,1)，所以陣列得多增加一個空間

## Code

```

class Solution:
 def uniquePathsWithObstacles(self, obstacleGrid: List[List[int]]) -> int:
 m, n = len(obstacleGrid), len(obstacleGrid[0])
 if m == 0 or n == 0 or obstacleGrid[0][0] == 1:
 return 0

 dp = [[0] * (n+1)] * (m+1)

 for i in range(1, m+1):
 for j in range(1, n+1):
 if i == 1 and j == 1:
 dp[i][j] = 1
 if obstacleGrid[i-1][j-1] == 1:
 dp[i][j] = 0
 else:
 dp[i][j] = dp[i-1][j] + dp[i][j-1]

 return dp[-1][-1]

```

## 64. Minimum Path Sum | 11/25

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

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

**Example:**

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

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

**Example:**

**Input:**

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

**Output:** 7

**Explanation:** Because the path 1→3→1→1→1 minimizes the sum.

### 思路

DP 思維:

初始化邊界

再寫dp推導式, 也就是把所有能知道的數字填上陣列

像這題, 能一路走到右 或 一路走到底, 就是已知, 須先初始化

### Code

```

class Solution:
 def minPathSum(self, grid: List[List[int]]) -> int:
 if not grid:
 return 0
 m, n = len(grid), len(grid[0])

 dp = [[0 for _ in range(n)] for _ in range(m)]

 dp[0][0] = grid[0][0]

 for i in range(1, m):
 dp[i][0] = dp[i-1][0] + grid[i][0]

 for j in range(1, n):
 dp[0][j] = dp[0][j-1] + grid[0][j]

 for i in range(1, len(grid)):
 for j in range(1, len(grid[0])):
 dp[i][j] = min(dp[i-1][j], dp[i][j-1]) + grid[i][j]

 return dp[-1][-1]

```

## 66. Plus One | 11/26

Given a non-empty array of digits representing a non-negative integer, plus one to the integer.

The digits are stored such that the most significant digit is at the head of the list, and each element in the array contain a single digit.

You may assume the integer does not contain any leading zero, except the number 0 itself.

Example 1:

Input: [1,2,3]

Output: [1,2,4]

Explanation: The array represents the integer 123.

Example 2:

Input: [4,3,2,1]

Output: [4,3,2,2]

Explanation: The array represents the integer 4321.

思路

Easy.

## Code

```
class Solution:
 def plusOne(self, digits: List[int]) -> List[int]:
 if digits[-1] < 9:
 digits[-1] += 1
 return digits

 else:
 carry = 1
 for i in range(len(digits)-1, -1, -1):
 if digits[i] + 1 > 9:
 digits[i] = 0
 carry = 1
 else:
 digits[i] += 1
 carry = 0
 break
 if carry == 1:
 digits.insert(0, 1)
 return digits
```

## 71. Simplify Path | 11/26

Given an absolute path for a file (Unix-style), simplify it. Or in other words, convert it to the canonical path.

In a UNIX-style file system, a period . refers to the current directory. Furthermore, a double period .. moves the directory up a level. For more information, see: Absolute path vs relative path in Linux/Unix

Note that the returned canonical path must always begin with a slash /, and there must be only a single slash / between two directory names. The last directory name (if it exists) must not end with a trailing /. Also, the canonical path must be the shortest string representing the absolute path.

### Example 1:

**Input:** "/home/"

**Output:** "/home"

**Explanation:** Note that there is no trailing slash after the last directory name.

### Example 2:

**Input:** "/../"

**Output:** "/"

**Explanation:** Going one level up from the root directory is a no-op, as the root level is the highest level you can go.

### Example 3:

**Input:** "/home//foo/"

**Output:** "/home/foo"

**Explanation:** In the canonical path, multiple consecutive slashes are replaced by a single one.

### Example 4:

**Input:** "/a/./b/../../c/"

**Output:** "/c"

## Example 5:

**Input:** "/a/.../..../b/.../c//.//"

**Output:** "/c"

## Example 6:

**Input:** "/a//b///c/d//././/..."

**Output:** "/a/b/c"

## 技巧

- join():
  - "".join()
  - 將後面的用split\_pattern分開
  - e.g.:
    - "-".join(['1','2','3','4'])
    - "1-2-3-4"
    - "".join(['1','2','3','4'])
    - "1234"

## 思路

使用 split('/'), 把每個item分隔出來

只取我們在意的, folder 及 ..

/folderA/.. 也就等於什麼都沒做, 因此可以把這個概念實踐:

1. 遇到folder就放進stack
2. 遇到 "..", 就把剛放進去的pop出 -> (什麼都沒做)

最後加上leading "/", 及使用"/" 分隔還在stack裡的元素即可

## Code

```
class Solution:
 def simplifyPath(self, path: str) -> str:
 stack = []
 for item in path.split('/'):
 # all we care is folder and ..
 # if "..", delete(pop) the last item push into stack
 if item not in ["", ".", ".."]:
 stack.append(item)

 if item == ".." and stack:
 stack.pop()
 return "/" + "/".join(stack)
```

---

## 72. Edit Distance | 11/26

---

Given two words word1 and word2, find the minimum number of operations required to convert word1 to word2.

You have the following 3 operations permitted on a word:

1. Insert a character
2. Delete a character
3. Replace a character

### **Example 1:**

**Input:** word1 = "horse", word2 = "ros"

**Output:** 3

**Explanation:**

horse → rorse (replace 'h' with 'r')

rorse → rose (remove 'r')

rose → ros (remove 'e')

### **Example 2:**

**Input:** word1 = "intention", word2 = "execution"

**Output:** 5

**Explanation:**

intention → inention (remove 't')

inention → enention (replace 'i' with 'e')

enention → exention (replace 'n' with 'x')

exention → exection (replace 'n' with 'c')

exection → execution (insert 'u')

思路

- X Hashmap → order is matter
- Delete, Insert, Remove.  
→ 沒有了規律可以說誰優  
“當下的決定對於之後的優劣，不知。”  
Given:  $\text{Word1}[i] \neq \text{Word2}[j]$

### 1. Insert.

insert  $\text{word2}[j]$  to  $\text{word1}[i]$ .  
→ next state, compare  $\langle \text{word1}[i], \text{word2}[j+1] \rangle$

### 2. Remove

remove  $\text{word1}[i]$ .  
→ next state, compare  $\langle \text{word1}[i+1], \text{word2}[j] \rangle$

### 3. Replace

replace to same.  
→ next state compare  $\langle \text{word1}[i+1], \text{word2}[j+1] \rangle$

這道題讓求從一個字符串轉變到另一個字符串需要的變換步驟，共有三種變換方式，插入一個字符，刪除一個字符，和替換一個字符。題目乍眼一看並不難，但是實際上卻暗藏玄機，對於兩個字符串的比較，一般都會考慮一下用 HashMap 統計字符出現的頻率，但是在這道題卻不可以這麼做，因為字符串的順序很重要。還有一種比較常見的錯誤，就是想當然的認為對於長度不同的兩個字符串，長度的差值都是要用插入操作，然後再對應每位字符，不同的地方用修改操作，但是其實這樣可能會多用操作，因為刪除操作有時同時可以達到修改的效果。比如題目中的例子 1，當把 horse 變為 rorse 之後，之後只要刪除第二個r，跟最後一個e，就可以變為 ros。實際上只要三步就完成了，因為刪除了某個字母後，原來左右不相連的字母現在就連一起了，有可能剛好組成了需要的字符串。所以在比較的時候，要嘗試三種操作，因為誰也不知道當前的操作會對後面產生什麼樣的影響。對於當前比較的兩個字符  $\text{word1}[i]$  和  $\text{word2}[j]$ ，若二者相同，一切好說，直接跳到下一個位置。若不相同，有三種處理方法，首先是直接插入一個  $\text{word2}[j]$ ，那麼  $\text{word2}[j]$  位置的字符就跳過了，接著比較  $\text{word1}[i]$  和  $\text{word2}[j+1]$  即可。第二個種方法是刪除，即將  $\text{word1}[i]$  字符直接刪掉，接著比較  $\text{word1}[i+1]$  和  $\text{word2}[j]$  即可。第三種則是將  $\text{word1}[i]$  修改

為  $\text{word2}[j]$ ，接著比較  $\text{word1}[i+1]$  和  $\text{word2}[j+1]$  即可。分析到這裡，就可以直接寫出遞歸的代碼，但是很可惜會 Time Limited Exceed，所以必須要優化時間複雜度，需要去掉大量的重複計算，這裡使用記憶數組  $\text{memo}$  來保存計算過的狀態，從而可以通過 OJ，注意這裡的  $\text{insertCnt}$ ,  $\text{deleteCnt}$ ,  $\text{replaceCnt}$  僅僅是表示當前對應的位置分別採用了插入，刪除，和替換操作，整體返回的最小距離，後面位置的還是會調用遞歸返回最小的

根據以往的經驗，對於字符串相關的題目且求極值的問題，十有八九都是用動態規劃 Dynamic Programming 來解，這道題也不例外。其實解法一的遞歸加記憶數組的方法也可以看作是 DP 的遞歸寫法。這裡需要維護一個二維的數組  $dp$ ，其大小為  $m \times n$ ， $m$  和  $n$  分別為  $word1$  和  $word2$  的長度。 $dp[i][j]$  表示從  $word1$  的前  $i$  個字符轉換到  $word2$  的前  $j$  個字符所需要的步驟。先給這個二維數組  $dp$  的第一行第一列賦值，這個很簡單，因為第一行和第一列對應的總有一個字符串是空串，於是轉換步驟完全是另一個字符串的長度。跟以往的 DP 題目類似，難點還是在於找出狀態轉移方程，可以舉個例子來看，比如  $word1$  是 "bbc"， $word2$  是 "abcd"，可以得到  $dp$  數組如下：

	$\emptyset$	a	b	c	d
$\emptyset$	0	1	2	3	4
b	1	1	1	2	3
b	2	2	1	2	3
c	3	3	2	1	2

通過觀察可以發現，當  $\text{word1}[i] == \text{word2}[j]$  時， $\text{dp}[i][j] = \text{dp}[i - 1][j - 1]$ ，其他情況時， $\text{dp}[i][j]$  是其左，左上，上的三個值中的最小值加1，其實這裡的左，上，和左上，分別對應的增加，刪除，修改操作，具體可以參見解法一種的講解部分，那麼可以得到狀態轉移方程為：

$$\begin{aligned} dp[i][j] = & \quad / \quad dp[i - 1][j - 1] && \text{if } word1[i - 1] == word2[j - 1] \\ & \backslash \quad \min(dp[i - 1][j - 1], \min(dp[i - 1][j], dp[i][j - 1])) + 1 && \text{else} \end{aligned}$$

## Code

## TLE解法, Intuitive?

```
self.minDistance(word1[1:], word2[1:]))
```

從Recursive轉換過來的DP, 參考圖

```
class Solution:
 def minDistance(self, word1: str, word2: str) -> int:
 m = len(word1)
 n = len(word2)

 # Initialize
 dp = [[0 for _ in range(n+1)] for _ in range(m+1)]
 dp[0][0] = 0

 for j in range (1,n+1):
 dp[0][j] = j

 for i in range (1, m+1):
 dp[i][0] = i

 # DP
 for i in range(1, m+1):
 for j in range(1, n+1):
 if word1[i-1] == word2[j-1]:
 dp[i][j] = dp[i-1][j-1]
 else:
 dp[i][j] = 1 + min(dp[i-1][j-1], dp[i][j-1], dp[i-1][j])

 return dp[-1][-1]
```

---

## 73. Set Matrix Zeroes | 11/27

---

Given a  $m \times n$  matrix, if an element is 0, set its entire row and column to 0. Do it in-place.

### **Example 1:**

**Input:**

```
[
 [1,1,1],
 [1,0,1],
 [1,1,1]
]
```

**Output:**

```
[
 [1,0,1],
 [0,0,0],
 [1,0,1]
]
```

### **Example 2:**

**Input:**

```
[
 [0,1,2,0],
 [3,4,5,2],
 [1,3,1,5]
]
```

**Output:**

```
[
 [0,0,0,0],
 [0,4,5,0],
 [0,3,1,0]
]
```

思路

空間複雜度  $mn$  的做法: 創一個等同大小的array, 並一個一個assign值

空間複雜度  $m+n$  的做法: 各創一個與行、列相同的array, 並只個別紀錄哪行哪列有出現0

## Code

```
class Solution:
 def setZeroes(self, matrix: List[List[int]]) -> None:
 """
 Do not return anything, modify matrix in-place instead.
 """

 m, n = len(matrix), len(matrix[0])
 row = [] #3
 col = [] #4

 for i in range(m):
 for j in range(n):
 if matrix[i][j] == 0:
 row.append(i)
 col.append(j)

 for zero_index in row:
 for j in range(n):
 matrix[zero_index][j] = 0

 for zero_index in col:
 for i in range(m):
 matrix[i][zero_index] = 0

 return matrix
```

---

## 1087. Brace Expansion | 12/1

---

A string S represents a list of words.

Each letter in the word has 1 or more options. If there is one option, the letter is represented as is. If there is more than one option, then curly braces delimit the options. For example, "`{a,b,c}`" represents options ["`a`", "`b`", "`c`"].

For example, "`{a,b,c}d{e,f}`" represents the list ["`ade`", "`adf`", "`bde`", "`bdf`", "`cde`", "`cdf`"].

Return all words that can be formed in this manner, in lexicographical order.

### Example 1:

**Input:** "{a,b}c{d,e}f"

**Output:** ["acdf", "acef", "bcdf", "bcef"]

### Example 2:

**Input:** "abcd"

**Output:** ["abcd"]

### Note:

1. `1 <= s.length <= 50`
2. There are no nested curly brackets.
3. All characters inside a pair of consecutive opening and ending curly brackets are different.

## 思路

Naive:

1. Make "`{a,b}c{d,e}f`" -> `[[a,b],[c],[d,e],[f]]`
2. DFS to find solution
3. Fail at "abcdefghijklmnpqrstuvwxyz" testcase

Wonderful Solution:

Using helper function to:

1. fetch out the elem in braces,
2. split out the string before braces
3. split out the string after braces

And recursively bind them in main function until no braces, e.g.:

1. ac{d,e}f
  - i. acdf -> return ["acdf"]
  - ii. acef -> return ["acef"]
2. bc{d,e}f

## Code

Wonderful Solution:

```
from functools import lru_cache
class Solution:
 def expand(self, S: str) -> List[str]:
 #self.helper.cache_clear()
 if '{' not in S:
 return [S]
 else:
 a, opts, b = self.helper(S)

 res = []
 for o in opts:
 res += self.expand(a+o+b)
 return res

 #@lru_cache(maxsize=None)
 def helper(self, s):
 i, j = s.index('{'), s.index('}')
 return s[:i], sorted(s[i+1:j].split(',')), s[j+1:]
```

Naive beat 6%:

```
class Solution:
 def expand(self, S: str) -> List[str]:
 def find_comb(outer, inner, comb):
 while len(comb) == n:
 ret.add("".join(comb[:]))
 return
 for i in range(outer, len(src)):
 for j in range(inner, len(src[i])):
 comb.append(src[i][j])
 find_comb(i+1, 0, comb)
 comb.pop()
 if '{' not in S:
 return [S]
 src, ret = [], set()
 i = 0
 while i < len(S):
 if S[i] == "{":
 br = []
```

```

 i += 1
 while S[i] != '}':
 if S[i] == ',':
 i += 1
 continue
 br.append(S[i])
 i += 1
 src.append(br)
 elif S[i] != "}" and S[i] != ",":
 src.append([S[i]])
 i += 1

n = len(src)
find_comb(0, 0, [])
return sorted(ret)

```

## 75. Sort Colors | 12/1

---

Given an array with n objects colored red, white or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

Note: You are not suppose to use the library's sort function for this problem.

Example:

Input: [2,0,2,1,1,0]

Output: [0,0,1,1,2,2]

Follow up:

A rather straight forward solution is a two-pass algorithm using counting sort.

First, iterate the array counting number of 0's, 1's, and 2's, then overwrite array with total number of 0's, then 1's and followed by 2's.

Could you come up with a one-pass algorithm using only constant space?

## 思路

如果要one path, 只能使用雙指針來搞

關鍵就是0一定在左, 2一定在右

所以可以使用兩個指針來記錄目前的最左跟最右在哪邊

遇到2就塞到目前的最右, 反之遇到0

注意while的check必須使用right而不是len(nums)

否則會把已經確定位置的2又換到錯的位置

## Code

```
class Solution:
 def sortColors(self, nums: List[int]) -> None:
 """
 Do not return anything, modify nums in-place instead.
 """

 # One Path:
 # self.sortColorsOptimap(nums)

 # Two Path:
 num_count_with_index = [0 for i in range(3)]
 for num in nums:
 num_count_with_index[num] += 1

 cur = 0
 for val, count in enumerate(num_count_with_index):
 for i in range(count):
 nums[cur+i] = val
 cur += count

 def sortColorsOptimap(self, nums):
 # One Path
 left, right = 0, len(nums)-1

 cur = 0
 while cur <= right: # Notice! Shouldn't use len(nums)
 if nums[cur] == 0:
 nums[cur], nums[left] = nums[left], nums[cur]
 cur += 1
 left += 1
 elif nums[cur] == 2:
 nums[cur], nums[right] = nums[right], nums[cur]
 right -= 1
 else:
 cur += 1
```

## 82. Remove Duplicates from Sorted List II | 12/2

---

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list.

Example 1:

Input: 1->2->3->3->4->4->5

Output: 1->2->5

Example 2:

Input: 1->1->1->2->3

Output: 2->3

### 思路

Iterative:

必須站在前一個節點往後看(並記錄這個節點為cur)

如果下兩個節點為一樣時, 進入刪除mode

使用while一直尋找到不同處

並把那個紀錄的cur與不同處連上

最後iterate prev, 就完成了

Recursive:

(no need the dummy node)

我們也可以使用遞歸來做，首先判空，如果 head 為空，直接返回。

然後判斷，若 head 之後的結點存在，且值相等，那麼先進行一個 while 循環，跳過後面所有值相等的結點，到最後一個值相等的點停下。

比如對於例子2來說，head 停在第三個結點1處，

然後對後面一個結點調用遞歸函數，即結點2，

這樣做的好處是，返回的值就完全把所有的結點都刪掉了。

若 head 之後的結點值不同，

那麼還是對 head 之後的結點調用遞歸函數，

將返回值連到 head 的後面，這樣 head 結點還是保留下來了，因為值不同嘛，最後返回 head 即可

## Code

Iterative:

```
class Solution:
 def deleteDuplicates(self, head: ListNode) -> ListNode:
 dum = ListNode(-1)
 dum.next = head
 prev = dum
 while prev.next and prev.next.next:
 cur = prev
 if prev.next.val == prev.next.next.val:
 while prev.next.next and prev.next.val == prev.next.next.val:
 prev = prev.next
 cur.next = prev.next.next
 prev = cur
 else:
 prev = prev.next
 return dum.next
```

Recursive:

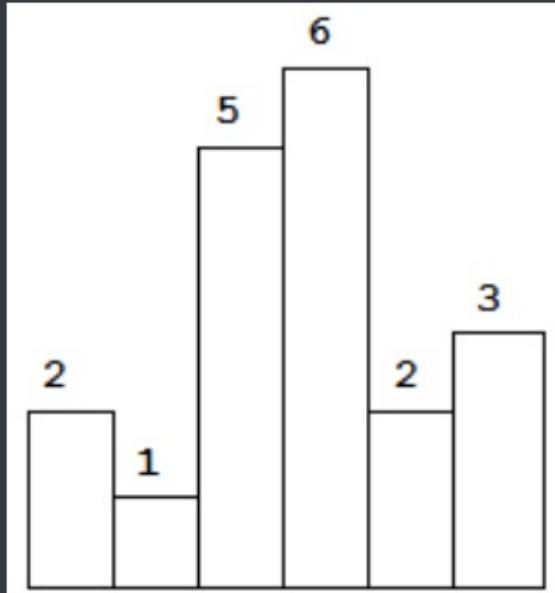
```
class Solution:
 def deleteDuplicates(self, head: ListNode) -> ListNode:
 if not head:
 return head

 if head.next and head.val == head.next.val:
 while head.next and head.val == head.next.val:
 head = head.next
 return self.deleteDuplicates(head.next)

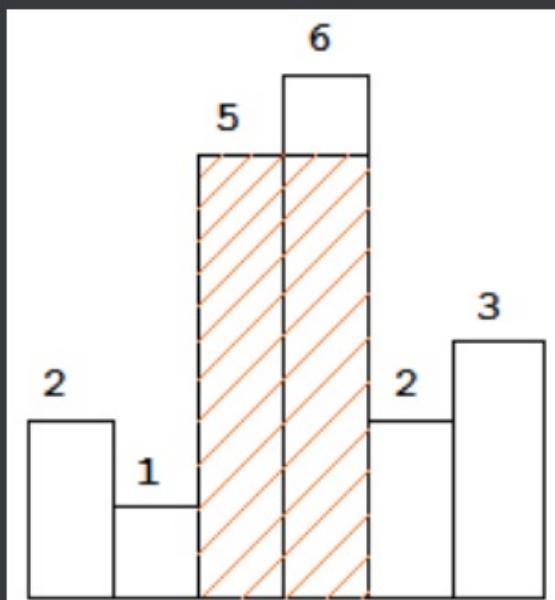
 head.next = self.deleteDuplicates(head.next) # 用這樣串起來的 !
 return head
```

## 84. Largest Rectangle in Histogram | 12/2

Given  $n$  non-negative integers representing the histogram's bar height where the width of each bar is 1, find the area of largest rectangle in the histogram.



Above is a histogram where width of each bar is 1, given height =  
[2, 1, 5, 6, 2, 3].



The largest rectangle is shown in the shaded area, which has area =  
10 unit.

## Example:

**Input:** [2,1,5,6,2,3]

**Output:** 10

## 技巧

mono-stack, increase-stack/decrease-stack:

- 自私的新元素會為了自己把所有造成破壞單調性的元素給踢除
- 解決問題的核心就在處理這塊，當前數字如果破壞了單調性，就會觸發處理棧頂元素的操作，而觸發數字有時候是解決問題的一部分
- 線性的時間複雜度是其最大的優勢，每個數字只進棧並處理一次
- 使用: 對於玩數組的題，如果相互之間關聯很大，那麼就可以考慮考慮單調棧能否解題。
  - 如: 使用local max來協助解題的題型
- 模板(以遞增stack舉例):
  - 常常會使用dummy elem協助解，如此題

```
def Solution(nums):
 for i in range(nums):
 while stack and heights[stack[-1]] > heights[i]: # trigger of process
 a = stack.pop()
 self.process()
 stack.append(i)

def process():
 # do sth
```

## Grandyang總結

## 思路

1. find Local max

每找到一個局部峰值（只要當前的數字大於後面的一個數字，那麼當前數字就看作一個局部峰值，跟前面的數字大小無關），然後向前遍歷所有的值，算出共同的矩形面積，每次對比保留最大值。這裡再說下為啥要從局部峰值處理，看題目中的例子，局部峰值為 2, 6, 3，我們只需在

這些局部峰值出進行處理，為啥不用在非局部峰值處統計呢，這是因為非局部峰值處的情況，後面的局部峰值都可以包括，比如1和5，由於局部峰值6是高於1和5的，所有1和5能組成的矩形，到6這裡都能組成，並且還可以加上6本身的一部分組成更大的矩形，那麼就不用費力氣去再統計一個1和5處能組成的矩形了。

## 2. 用mono-stack

那麼既然需要用單調棧來做，首先要考慮到底用遞增棧，還是用遞減棧來做。我們想啊，遞增棧是維護遞增的順序，當遇到小於棧頂元素的數就開始處理，而遞減棧正好相反，維護遞減的順序，當遇到大於棧頂元素的數開始處理。那麼根據這道題的特點，我們需要按從高板子到低板子的順序處理，先處理最高的板子，寬度為1，然後再處理旁邊矮一些的板子，此時長度為2，因為之前的高板子可組成矮板子的矩形，因此我們需要一個遞增棧，當遇到大的數字直接進棧，而當遇到小於棧頂元素的數字時，就要取出棧頂元素進行處理了，那取出的順序就是從高板子到矮板子了，於是乎遇到的較小的數字只是一個觸發，表示現在需要開始計算矩形面積了，為了使得最後一塊板子也被處理，這裡用了個小 trick，在高度數組最後面加上一個0，這樣原先的最後一個板子也可以被處理了。

## Code

find local max:

```
class Solution:
 def largestRectangleArea(self, heights: List[int]) -> int:
 # find local max: larger than the next elem
 # don't need to consider the previous elem, because you walk from there
 res = 0
 for i in range(len(heights)):
 if i+1 < len(heights) and heights[i] <= heights[i+1]:
 continue # Use continue here, because we need to consider the last

 minH = heights[i]
 for j in range(i, -1, -1):
 minH = min(minH, heights[j])
 area = minH * (i-j+1)
 res = max(res, area)
 return res
```

Using mono-stack(increased stack):

```
class Solution:
 def largestRectangleArea(self, heights: List[int]) -> int:
 # find local max: larger than the next elem
 # don't need to consider the previous elem, because you walk from there
```

```

res = 0
trick: need to consider the last elem, push 0 (dummy) to the last
Also push -1 in stack to indicate the last elem in heights(dummy), height
stack = [-1]
heights.append(0)

for i in range(len(heights)):
 while stack and heights[stack[-1]] > heights[i]:
 h = heights[stack.pop()]
 res = self.process(res, h, i, stack[-1])
 # h = heights[stack.pop()]
 # w = i - stack[-1] - 1 # stack[-1] is the height before the one j
 # max_a = max(max_a, h*w)
 stack.append(i)
return res

def process(self, res, height, i, j):
 res = max(res, height * (i-j-1))
 return res

```

## 86. Partition List | 12/3

Given a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x.

You should preserve the original relative order of the nodes in each of the two partitions.

Example:

Input: head = 1->4->3->2->5->2, x = 3

Output: 1->2->2->4->3->5

這道題要求我們劃分鏈表，把所有小於給定值的節點都移到前面，大於該值的節點順序不變，相當於一個局部排序的問題。那麼可以想到的一種解法是首先找到第一個大於或等於給定值的節點，用題目中給的例子來說就是先找到4，然後再找小於3的值

## 技巧

在操作list時，

如果head的位置matter的話，

盡量使用cursor去操作list(以免誤動到head了)

像是方法中使用cur, new\_cur去操作

而dum, new\_dum永遠維持不變

## 思路

分割時,可以真的分割如1. 或者new一個新list來代表分割後的結果

1. 直接斷點法: (複雜)

斷點斷在第一個大於x的點,

分成:前半部(break\_p), 後半部(new\_head)

並將所有後半部mis-positioned的點放到break\_p之後

2. (Better) 直接new一個新的list法:

直接在走訪時操作, 連接到新的list後面

走訪完後, 把兩個list連在一起

## Code

新list法:

```
class Solution:
 def partition(self, head: ListNode, x: int) -> ListNode:
 dum, new_dum = ListNode(-1), ListNode(-1)
 dum.next = head

 # Assign the cursor to do the operation
 # => dum, new_dum never move!!
 cur, new_cur = dum, new_dum

 while cur.next:
 if cur.next.val < x:
 new_cur.next = cur.next
 cur.next = cur.next.next

 # move the new_dum cursor
 new_cur = new_cur.next
 else:
 cur = cur.next

 # Connect two list together, Note: still use the cursor to do the op
```

```
new_cur.next = dum.next
return new_dum.next
```

斷點法：

```
class Solution:
 def partition(self, head: ListNode, x: int) -> ListNode:
 dum = ListNode(-1)
 dum.next = head
 prev = dum
 break_p, new_head = None, None

 # find the breaking point, and break the list into two
 while prev.next:
 if prev.next.val >= x:
 break_p = prev
 new_head = prev.next
 break
 else:
 prev = prev.next
 self.arrange(break_p, new_head, x)
 return dum.next

 def arrange(self, break_p, new_head, x):
 # link the misposition node in new_head list to break_p
 while new_head:
 if new_head.next and new_head.next.val < x:
 # keep the status
 b_next = break_p.next
 n_next = new_head.next.next

 # link that node to break_p
 break_p.next = new_head.next
 break_p.next.next = b_next
 break_p = break_p.next # don't forget to move forward the break_p p

 # fill the removed position
 new_head.next = n_next
 else:
 new_head = new_head.next
```

## 85. Maximal Rectangle | 12/4

Given a 2D binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return its area.

Example:

**Input:**

[

["1","0","1","0","0"],  
["1","0","1","1","1"],  
["1","1","1","1","1"],  
["1","0","0","1","0"]

]

**Output:** 6

## 思路

之前找histogram 長方形最大值的延伸版

轉換方法是把每一行看作新的一層，並往上累加高度

累加高度的方法是：

對於每一個點，如果是‘0’，則賦0，如果是‘1’，就賦之前的height值加上1

## Code

```
class Solution(object):
 def maximalRectangle(self, matrix):
 """
 :type matrix: List[List[str]]
 :rtype: int
 """
 if not matrix: return 0
```

```

row, col = len(matrix), len(matrix[0])
res, heights = 0, [0 for _ in range(col)]

for i in range(row):
 for j in range(col):
 heights[j] = (1+heights[j]) if matrix[i][j] != '0' else 0
 res = max(res, self.largestRectangleArea(heights))
return res

def largestRectangleArea(self, heights):
 # [3, 1, 3, 2, 2]
 heights = list(map(int, heights))
 max_a = 0
 stack = [-1]
 heights.append(0)
 for i in range(len(heights)):
 if heights[i] < heights[stack[-1]]:
 while stack and heights[i] < heights[stack[-1]]:
 h = heights[stack.pop()]
 w = i - stack[-1] - 1
 max_a = max(max_a, h*w)
 stack.append(i)
 return max_a

```

## 88. Merge Sorted Array | 12/5

---

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

Note:

The number of elements initialized in nums1 and nums2 are m and n respectively.

You may assume that nums1 has enough space (size that is greater or equal to m + n) to hold additional elements from nums2.

Example:

Input:

nums1 = [1,2,3,0,0,0], m = 3

nums2 = [2,5,6], n = 3

Output: [1,2,2,3,5,6]

## 思路

因為最終size是固定的,

因此從後往前比, 大的就放在最後面

## Code

```
class Solution(object):
 def merge(self, nums1, m, nums2, n):
 while m > 0 and n > 0:
 if nums2[n-1] > nums1[m-1]:
 nums1[m+n-1] = nums2[n-1]
 n -= 1
 else:
 nums1[m+n-1] = nums1[m-1]
 m -= 1
 if n > 0:
 nums1[:n] = nums2[:n]
```

精簡版, 加上要更新要把nums2放到nums1後的條件

```
class Solution(object):
 def merge(self, nums1, m, nums2, n):
 while n > 0:
 if m <= 0 or nums2[n-1] > nums1[m-1]:
 nums1[m+n-1] = nums2[n-1]
 n -= 1
 else:
 nums1[m+n-1] = nums1[m-1]
 m -= 1
```

## 97. Interleaving String | 12/5

Given s1, s2, s3, find whether s3 is formed by the interleaving of s1 and s2.

Example 1:

Input: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbcbcac"

Output: true

Example 2:

Input: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbbaccc"

Output: false

## 學習

1. Recursive 思路漏洞:

第一次的寫法:

```
if m > 0 and s1[0] == s3[0]:
 return self.isInterleave(s1[1:], s2, s3[1:])
if n > 0 and s1[0] == s3[0]:
 return self.isInterleave(s1, s2[1:], s3[1:])
```

並考慮這個case:

```
s1 = aabbc
s2 = abbdc
s3 = abbbc
```

這種方式會造成s1與s3匹配過後發現沒有路了return False

然後就一直return False回上層導致s2與s3沒有機會嘗試到

2. 在用地迴時思考有沒有任何可能可以記錄重複的步驟以至於能優化，

像這題就是以紀錄兩個字串剩餘的長度來達成跳過重複

## 思路

就是遞迴逐一比對, 注意recursive的思路, 以及如何優化遞迴

## Code

在大數據時會TLE

```

class Solution:
 def isInterleave(self, s1: str, s2: str, s3: str) -> bool:
 m, n, mn = len(s1), len(s2), len(s3)
 if m + n != mn:
 return False
 if m == 0 and n == 0 and mn == 0:
 return True

 if m > 0 and s1[0] == s3[0]:
 if self.isInterleave(s1[1:], s2, s3[1:]):
 return True
 if n > 0 and s2[0] == s3[0]:
 if self.isInterleave(s1, s2[1:], s3[1:]):
 return True
 return False

```

優化, cache紀錄path

```

class Solution:
 def __init__(self):
 self.visited = set()

 def isInterleave(self, s1: str, s2: str, s3: str) -> bool:
 m, n, mn = len(s1), len(s2), len(s3)
 if m + n != mn:
 return False
 if m == 0 and n == 0 and mn == 0:
 return True

 if m > 0 and s1[0] == s3[0]:
 if (m-1, n) not in self.visited:
 self.visited.add((m-1, n))
 if self.isInterleave(s1[1:], s2, s3[1:]):
 return True
 if n > 0 and s2[0] == s3[0]:
 if (m, n-1) not in self.visited:
 self.visited.add((m, n-1))
 if self.isInterleave(s1, s2[1:], s3[1:]):
 return True
 return False

```

不使用遞迴, 用stack:

```

def isInterleave4(self, s1, s2, s3):
 r, c, l= len(s1), len(s2), len(s3)
 if r+c != l:
 return False
 stack, visited = [(0, 0)], set((0, 0))

```

```

while stack:
 x, y = stack.pop()
 if x+y == l:
 return True
 if x+1 <= r and s1[x] == s3[x+y] and (x+1, y) not in visited:
 stack.append((x+1, y)); visited.add((x+1, y))
 if y+1 <= c and s2[y] == s3[x+y] and (x, y+1) not in visited:
 stack.append((x, y+1)); visited.add((x, y+1))
return False

```

## 103. Binary Tree Zigzag Level Order Traversal | 12/5

Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).

For example:

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

```

3
/
9 20
/ \
15 7

```

return its zigzag level order traversal as:

```

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

### 技巧

層走訪:創空間+層參數

### 思路

正確的level order走訪:

在push queue的時候多push level的參數,

並用level來決定是否要new一個新的room給this level

## Code

```
class Solution:
 def zigzagLevelOrder(self, root: TreeNode) -> List[List[int]]:
 if not root:
 return []
 flag, queue, res = 0, [], []
 queue.append(root);
 res.append([root.val])

 while queue:
 flag += 1 # flag to reverse the level_val
 level, level_val = [], []
 while queue: # to confine in this level
 top = queue.pop(0)
 if top.left:
 level.append(top.left)
 if top.right:
 level.append(top.right)

 while level: # repush into the queue for next level
 top = level.pop(0)
 level_val.append(top.val)
 queue.append(top)

 if flag%2 != 0:
 level_val = level_val[::-1]

 if len(level_val) != 0:
 res.append(level_val)
 return res
```

level order精簡版:

```
class Solution:
 def zigzagLevelOrder(self, root: TreeNode) -> List[List[int]]:
 queue, res = [], []
 queue.append((root, 0))

 while queue:
 top, level = queue.pop(0)
 if top:
```

```
if len(res) < level+1: # new a room for this level
 res.append([])

if level % 2 == 0:
 res[level].append(top.val)
else:
 res[level].insert(0,top.val)
queue.append((top.left, level+1))
queue.append((top.right, level+1))
return res
```

---