**LeetCode 277 Find celebrity**

**提供你一個know（int a, int b），a認識b，就會返回true**

**明星嚴格定義是剩餘的所有人認識他，但是他不認識其餘的人**

**給你n個人，從1號到n號，找出明星是哪個號，最多只有一個明星，最少沒有**

**思路：頭尾雙指針，直到只剩下一個人，i號就有可能是明星。之後對i號問下其餘人是否認識他，他是否認識其餘人**

**bool** knows(**int** a, **int** b);

**class** Solution {

**public**:

**int** findCelebrity(**int** n) {

**int** left = 0;

**int** right = n - 1;

**while** (left < right) {

**if**(knows(left, right)) {

left ++;

} **else** {

right --;

}

}

**for** (**int** i = 0; i < n; i ++) {

**if** (i==left || (knows(i, left) && !knows(left, i))) {

**continue**;

} **else** {

**return** -1;

}

}

**return** left;

}

};

[**leetcode[161] One Edit Distance**](http://www.cnblogs.com/higerzhang/p/4185887.html)

判断两个字符串的编辑距离是不是1. 對字符串的操作可以使1. 刪除一個字符，2添加一個字符 3替換一個字符

// Returns true if edit distance between s1 and

// s2 is one, else false

bool isEditDistanceOne(string s1, string s2)

{

// Find lengths of given strings

int m = s1.length(), n = s2.length();

// If difference between lengths is more than

// 1, then strings can't be at one distance

if (abs(m - n) > 1)

return false;

int count = 0; // Count of edits

int i = 0, j = 0;

while (i < m && j < n)

{

// If current characters don't match

if (s1[i] != s2[j])

{

if (count == 1)

return false;

// If length of one string is

// more, then only possible edit

// is to remove a character

if (m > n)

i++;

else if (m< n)

j++;

else //Iflengths of both strings is same

{

i++;

j++;

}

// Increment count of edits

count++;

}

else // If current characters match

{

i++;

j++;

}

}

// If last character is extra in any string

if (i < m || j < n)

count++;

return count == 1;

}

[**LeetCode 314. Binary Tree Vertical Order Traversal（二叉树垂直遍历）**](http://blog.csdn.net/jmspan/article/details/51216866)

思路：按層遍歷。加一個類position，有treenode和int pos組成。root的pos是0，左子樹減1，右子樹加1。把treenode合成為position類，放在hashmap<Integer, List<position>>中

1. **public** **class** Solution {
2. **public** List<List<Integer>> verticalOrder(TreeNode root) {
3. List<List<Integer>> results = **new** ArrayList<>();
4. **if** (root == **null**) **return** results;
5. **int** min = Integer.MAX\_VALUE;
6. **int** max = Integer.MIN\_VALUE;
7. Map<Integer, List<Integer>> map = **new** HashMap<>();
8. LinkedList<Position> queue = **new** LinkedList<>();
9. queue.add(**new** Position(root, 0));
10. **while** (!queue.isEmpty()) {
11. Position position = queue.remove();
12. min = Math.min(min, position.column);
13. max = Math.max(max, position.column);
14. List<Integer> list = map.get(position.column);
15. **if** (list == **null**) {
16. list = **new** ArrayList<>();
17. map.put(position.column, list);
18. }
19. list.add(position.node.val);
20. **if** (position.node.left != **null**) queue.add(**new** Position(position.node.left, position.column-1));
21. **if** (position.node.right != **null**) queue.add(**new** Position(position.node.right, position.column+1));
22. }
23. **for**(**int** i=min; i<=max; i++) {
24. List<Integer> list = map.get(i);
25. **if** (list != **null**) results.add(list);
26. }
27. **return** results;
28. }
29. }
30. **class** Position {
31. TreeNode node;
32. **int** column;
33. Position(TreeNode node, **int** column) {
34. **this**.node = node;
35. **this**.column = column;
36. }
37. }

75. Sort Colors

思路：左邊指針指著第一個非零位置，右邊指針指著非2的位置。

找第三個個指針從左到右搜一遍，如果找到0，與左邊交換，然後左邊指針向右，第三個指針也向右。

如果搜到2，與右邊指針交換，右邊指針向左。第三指針不動，因為有可能交換后，第三指針指著0，需要跟左邊的再交換。

第三指針遇到1不管，跳過。

public class Solution {

public void sortColors(int[] nums) {

int low=0;

int high=nums.length-1;

int i=0;

while(i<=high){

if(nums[i]==0){

swap(nums,i,low);

i++;

low++;

}

else if(nums[i]==2){

swap(nums,i,high);

high--;

}

else{

i++;

}

}

}

private void swap(int[] nums, int i1, int i2){

int tmp = nums[i1];

nums[i1] = nums[i2];

nums[i2] = tmp;

}

}

49. Group Anagrams

Given an array of strings, group anagrams together.

For example, given: ["eat", "tea", "tan", "ate", "nat", "bat"],   
Return:

[

["ate", "eat","tea"],

["nat","tan"],

["bat"]

]

public static List<List<String>> groupAnagrams(String[] strs) {

int[] prime = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103};//最多10609个z

List<List<String>> res = new ArrayList<>();

HashMap<Integer, Integer> map = new HashMap<>();

for (String s : strs) {

int key = 1;

for (char c : s.toCharArray()) {

key \*= prime[c - 'a'];

}

List<String> t;

if (map.containsKey(key)) {

t = res.get(map.get(key));

} else {

t = new ArrayList<>();

res.add(t);

map.put(key, res.size() - 1);

}

t.add(s);

}

return res;

}

79. Word Search

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.

For example,  
Given **board** =

[

['A','B','C','E'],

['S','F','C','S'],

['A','D','E','E']

]

**word** = "ABCCED", -> returns true,  
**word** = "SEE", -> returns true,  
**word** = "ABCB", -> returns false.

思路：暴力法，沒其他辦法，用一個visited[][]記錄走過哪裡

public class Solution {

static boolean[][] visited;

public boolean exist(char[][] board, String word) {

visited = new boolean[board.length][board[0].length];

for(int i = 0; i < board.length; i++){

for(int j = 0; j < board[i].length; j++){

if((word.charAt(0) == board[i][j]) && search(board, word, i, j, 0)){

return true;

}

}

}

return false;

}

private boolean search(char[][]board, String word, int i, int j, int index){

if(index == word.length()){

return true;

}

if(i >= board.length || i < 0 || j >= board[i].length || j < 0 || board[i][j] != word.charAt(index) || visited[i][j]){

return false;

}

visited[i][j] = true;

if(search(board, word, i-1, j, index+1) ||

search(board, word, i+1, j, index+1) ||

search(board, word, i, j-1, index+1) ||

search(board, word, i, j+1, index+1)){

return true;

}

visited[i][j] = false;

return false;

}

}

209. Minimum Size Subarray Sum

Given an array of **n** positive integers and a positive integer **s**, find the minimal length of a **contiguous** subarray of which the sum ≥ **s**. If there isn't one, return 0 instead.

For example, given the array [2,3,1,2,4,3] and s = 7,  
the subarray [4,3] has the minimal length under the problem constraint.

public class Solution {

public int minSubArrayLen(int s, int[] nums) {

int size=Integer.MAX\_VALUE;

int left=0;

int sum=0;

for(int i=0;i<nums.length;i++){

sum+=nums[i];

while(left<=i&&sum>=s){

size=Math.min(size,i-left+1);

sum-=nums[left];

left++;

}

}

return size==Integer.MAX\_VALUE?0:size;

}

}

71. Simplify Path

当遇到“/../"则需要返回上级目录，需检查上级目录是否为空。

当遇到"/./"则表示是本级目录，无需做任何特殊操作。

当遇到"//"则表示是本级目录，无需做任何操作。

当遇到其他字符则表示是文件夹名，无需简化。

当字符串是空或者遇到”/../”，则需要返回一个"/"。

当遇见"/a//b"，则需要简化为"/a/b"。

思路：使用stack，遍歷玩字符串數組后，把stack翻轉過來，放到StringBuilder里，如果最後StringBuilder是空，就返回一個“/”

public String simplifyPath(String path) {

if(path == null||path.length()==0)

return path;

Stack<String> stack = new Stack<String>();

String[] list = path.split("/");

for(int i=0; i<list.length; i++){

if(list[i].equals(".")||list[i].length()==0)

continue;

else if(!list[i].equals(".."))

stack.push(list[i]);

else{

if(!stack.isEmpty())

stack.pop();

}

}

StringBuilder res = new StringBuilder();

Stack<String> temp = new Stack<String>();

while(!stack.isEmpty())

temp.push(stack.pop());

while(!temp.isEmpty())

res.append("/"+temp.pop());

if(res.length()==0)

res.append("/");

return res.toString();

}

# [[LeetCode] Walls and Gates 墙和门](http://www.cnblogs.com/grandyang/p/5285868.html)

You are given a m x n 2D grid initialized with these three possible values.

1. -1 - A wall or an obstacle.
2. 0 - A gate.
3. INF - Infinity means an empty room. We use the value 231 - 1 = 2147483647 to represent INF as you may assume that the distance to a gate is less than 2147483647.

Fill each empty room with the distance to its nearest gate. If it is impossible to reach a gate, it should be filled with INF.

For example, given the 2D grid:

INF -1 0 INF

INF INF INF -1

INF -1 INF -1

0 -1 INF INF

After running your function, the 2D grid should be:

3 -1 0 1

2 2 1 -1

1 -1 2 -1

0 -1 3 4

这道题类似一种迷宫问题，规定了-1表示墙，0表示门，让求每个点到门的最近的曼哈顿距离，这其实类似于求距离场Distance Map的问题，那么我们先考虑用DFS来解，思路是，我们搜索0的位置，每找到一个0，以其周围四个相邻点为起点，开始DFS遍历，并带入深度值1，如果遇到的值大于当前深度值，我们将位置值赋为当前深度值，并对当前点的四个相邻点开始DFS遍历，注意此时深度值需要加1，这样遍历完成后，所有的位置就被正确地更新了，参见代码如下：

解法一：

[复制代码](javascript:void(0);)

class Solution {

public:

void wallsAndGates(vector<vector<int>>& rooms) {

for (int i = 0; i < rooms.size(); ++i) {

for (int j = 0; j < rooms[i].size(); ++j) {

if (rooms[i][j] == 0) {

dfs(rooms, i + 1, j, 1);

dfs(rooms, i - 1, j, 1);

dfs(rooms, i, j + 1, 1);

dfs(rooms, i, j - 1, 1);

}

}

}

}

void dfs(vector<vector<int>> &rooms, int i, int j, int val) {

if (i < 0 || i >= rooms.size() || j < 0 || j >= rooms[i].size()) return;

if (rooms[i][j] > val) {

rooms[i][j] = val;

dfs(rooms, i + 1, j, val + 1);

dfs(rooms, i - 1, j, val + 1);

dfs(rooms, i, j + 1, val + 1);

dfs(rooms, i, j - 1, val + 1);

}

}

};

[复制代码](javascript:void(0);)

Reservoir Sampling從給定範圍中隨機抽取K個樣本的算法

現在0到K位置放入k個值，之後用來打亂。

for(int i = k + 1; i  <= N; ++i)

{

    m = random(1, i);

    if(k >= m)

        swap the mth value with the ith value;

}

334. Increasing Triplet Subsequence

數組中存不存在長度為3的遞增sequence

思路：維護兩個val, min和secondmin，開始時兩個都設為Interger.max

public class Solution {

public boolean increasingTriplet(int[] nums) {

if (nums == null || nums.length < 3 ){

return false;

}

int min = Integer.MAX\_VALUE;

int secondMin = Integer.MAX\_VALUE;

for (int i = 0; i < nums.length; i++){

if (nums[i] <= min){

min = nums[i];

}

else if (nums[i] <= secondMin){

secondMin = nums[i];

}

else {

return true;

}

}

return false;

}

}