**Leet code刷題紀錄**

2020/06/30

**665. Non-decreasing Array**

沒寫出來

遍歷個元素然後更改值正確。但少考慮何時要修改當前值，何時修改後一值(此關鍵為看當前值與前一值之比較)

2020/-7/01

**633. Sum of Square Numbers**

如果用兩個迴圈會exceed time limit

只能用一個迴圈

最多人使用兩pointer一等於0 一等於sqrt©然後往中間靠

2020/07/04

**154. Find Minimum in Rotated Sorted Array II**

因為有duplicate，所以如果nums[mid] = nums[r]會無法判斷旋轉斷點在哪裡，這時候可以左移一格r

**540. Single Element in a Sorted Array**

可以用^ xor 運算子減少程式碼 nums[mid]==nums[mid^1] 來判斷單一元素在左方或右方

2020/07/05

**4. Median of Two Sorted Arrays**

尋找第(m+n+1)/2與(m+n+2)/2的平均。尋找第k個元素用一recursive function實現，利用二分搜尋法比較num1與num2的第k/2元素。若num1的第k/2元素比較小，則說明第k元素不可能位於num1第k/2元素前，固可將指針增加k/2，並進入下一次遞迴。反之亦然。若其中一array已被搜索完，則直接返回另外一array之第k個元素。若k值等於1，則返回num1及num2兩個初始指針指向之值中的min。

2020/07/06

**O(n logn)：合併排序 (Merge Sort)**

**拆分**

把大陣列切一半成為兩個小陣列

把切好的兩個小陣列再各自切一半

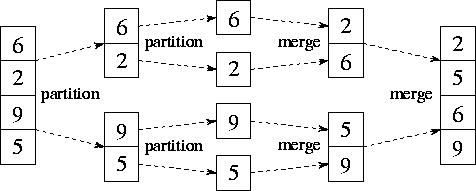
重複步驟二直到每個小陣列都只剩一個元素

**合併**

排序兩個只剩一個元素的小陣列並合併

把兩邊排序好的小陣列合併並排序成一個陣列

重複步驟二直到所有小陣列都合併成一個大陣列



2020/07/07

**75. Sort Colors**

為達到one pass，除了遍歷以外要有兩個指針(一紅指向0，一藍指向2)，red一開始指向頭，當遍歷遇到0時則將此數字與red數字交換，交換玩red指針與i +1；當遍歷遇到2時則將此數字與blue數字交換，叫喚玩blue指針-1；當遇遍歷遇到1時則部動作i+1。

2020/07/08

在Priority queue是以heap實作。**若priority queue內之元素是pair，則以pair中第一個元素之大小作為queue pop的比較依據**

**451. Sort Characters By Frequency**

先用map紀錄出現字數，再放入priority queue。

(如果用內建sort的話可以直接對input sort，但自寫sort function與考慮到若出現次數相同的情況)

**47. Permutations II**

麻煩

2020/07/15

**310. Minimum Height Trees**

原本想法:遍歷每個點然後算出以每個點為root時之樹高，記錄最小樹高以及其root。問題: 跑testcase時會超過time limit，而且若邊沒有出現的數字時樹高為0，會直接變答案，錯誤。

解答建議方式:**利用剝洋蔥法，一層層脫去葉節點，剩下的一個或兩個節點就是答案**。

2020/07/21

**343. Integer Break**

利用dp解。Dp[i]為可得知最大乘積。第一個外圈計算遍歷dp[i]，第二圈內圈計算數字全部拆法(此方法為O(n2))。另外利用找規律(拆出3)

**646. Maximum Length of Pair Chain**

好像不用dp比較快

2020/07/22

**494. Target Sum**

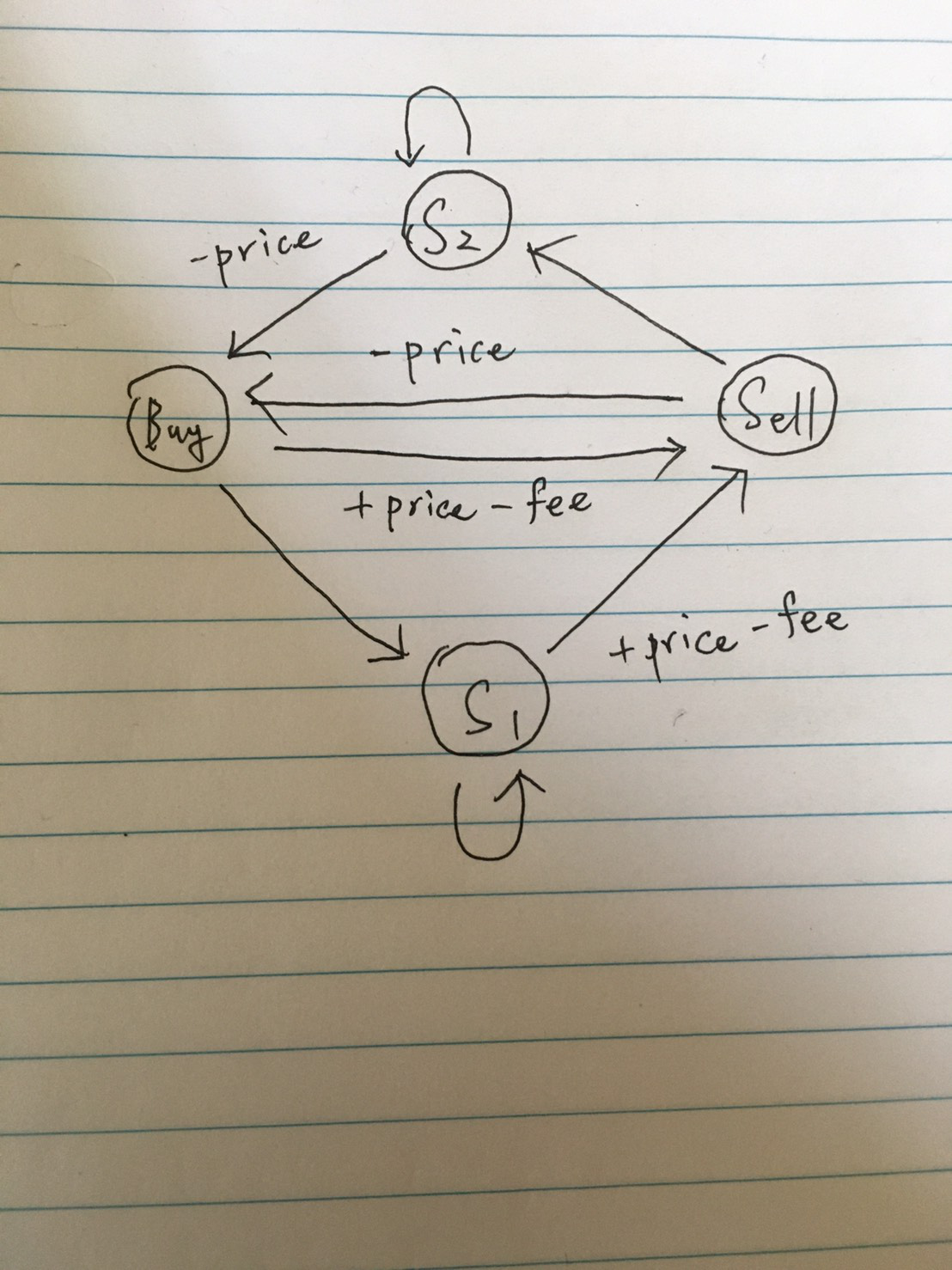
有用recursive function 解出來

其實可以用dp解

2020/07/23

**714. Best Time to Buy and Sell Stock with Transaction Fee**

畫finite state machine解dp



**932. Beautiful Array**

略難，思路:

因为2\*A[k]是偶数，如果要求2\*A[K]!=A[I]+A[J]那么可以构造位置在A[k]左边的全部放奇数，位置在A[k]右边的全部放偶数。这样就保证了对于K位置而言，这个性质是满足的。

然後就從{1}開始擴大Beautiful array，左半邊 x 2 – 1，右半邊 x 2。

**312. Burst Balloons**

要利用 dp 解，暴力遍歷recursion會超過time limit。d[i][j]為在i ~ j 區間可以得到的最多金幣，然後輪流把I ~J 間的氣球當作最後一個打爆的氣球，看哪一個最後打爆可以獲得最多金幣並更新。Dp遞迴關係式:

dp[left][right] = max(dp[left][right], nums[left - 1] \* nums[i] \* nums[right + 1] + dp[left][i - 1] + dp[i + 1][right])

注意: dp 便利要從長度為一的dp[1][1]，dp[2][2]…開始，到長度為二長度為三依序下去。

2020/07/27

**462. Minimum Moves to Equal Array Elements II**

想像把數字畫在數線上，若數字為奇數個，則最短距離數為移動到中間的那個數上；若為偶數個，則最短距離為移動到最中間兩個數之間的區間之任一數。總步數算法為兩端數字為兩兩一對，其距離的數字總和。

\_\_\_\_\_\_C\_\_\_\_\_A\_\_\_\_\_P\_\_\_\_\_\_\_B\_\_\_\_\_\_D\_\_\_\_\_\_

移動距離合為，|D – C| + |B – A|。

**169. Majority Element**

原本方法是先sorting 後(O(nlogn)再累加當前數字，若超過半數則返回答案，若沒有則換下一個數字。

Optimal solution: Boyer-Moore Majority Vote可O(n) 時間 O(1)空間

这种投票法先将第一个数字假设为过半数，然后把计数器设为1，比较下一个数和此数是否相等，若相等则计数器加一，反之减一。然后看此时计数器的值，若为零，则将下一个值设为候选过半数。以此类推直到遍历完整个数组，当前候选过半数即为该数组的过半数。

2020/07/30

在array裡搜尋missing element或重複element時，可以試看使用**二進位either or ^ 的性質，x ^ 0s = x , x ^ 1s = ~x, x ^ x =0。**

**476. Number Complement**

注意在使用 << 1 這種方式時，有可能會overflow。

**260. Single Number III**

將所有數字either or 後的數字為兩個single number的 either or，並找出此數字第一位為一的數字(用diff ^= -diff) 用這個數字來對數字分類必可分出此二數字。

2020/08/03

**218. The Skyline Problem**

跳過 ㄏㄏ

2020/08/04

map:

map.find() 時間為O(1)，若沒有找到東西則回傳map.end()\* 回傳map的最後一個元素的下一個，這個元素不存在，通常用於表示資料已經到達結尾。

Hash map vs map: 實現方法不同(hash table vs binary tree)

**Hash map**

Pro: 收尋容易

Con: 刪除新增麻煩

**Map**

Pro: 刪除新增容易

Con: 收尋麻煩

2020/08/05

**503. Next Greater Element II**

使用單調棧(simple stack)。因為此為環形array，所以先將0 ~ (n-2)之元素複製加在nums後面。同時先把n-2 ~ 0的元素先放入stack中。接著開始比較，若current element >= stack.top()則 pop()， < 則為next greater number。(自己想ㄉ) 網上解答為travelsal nums兩遍 然後做大同小異的事。

**217. Contains Duplicate**

用unordered\_set 若沒看過就放入set中，有看過則return true。也可排序後，看左右是否有相同，好處為space complexity 為 O(1)，壞處為sort會增加時間。

2020/08/07

**697. Degree of an Array**

原本思路是先遍歷一次找出各數字出現的次數，與各數字第一次出現與最好一次出現，再用最大degree中找出最短子array。**網路最好解法(一次遍歷)**：用兩個map紀錄數字出現數字與首次出現位置，並用當前位置減去首次出現位置即可找出子array

**594. Longest Harmonious Subsequence**

我用hash map先第一次loop紀錄每個數字出現幾次，在第二次loop然後每次看key + 1存不存在，如果存在的話把出現次數加上key出現次數，然後loop出最大值。網路最好解法(一次遍歷): 用一次loop 將map存入map中，同時觀察key + 1 與key – 1存不存在，存在則比較有沒有比較大的，邊加入map邊更新。

2020/08/08

**287. Find the Duplicate Number**

網路詳解: **利用快慢指針的概念(fast = nums[nums[fast]], slow = nums[slow])其duplicate number為環的起點。**所以第一個while當fast = slow 時，起點到相遇點 = 環長度，所以這時再把快指針移回原點並改成慢速，再次相遇時即為環點(duplicate number)。※去複習Linked List Cycle

2020/08/10

**313. Super Ugly Number**

因super ugly number都由質數表primes來的。所以下一個super ugly number一定由現有的super number \* prime 而來。故設立一ptr vector，紀錄每一個prime已經乘到哪一個super ugly number。找下一個super ugly number的方式就是每個prime \* 此prime還沒乘過的super ugly number，然後比較找最小值，此值就是next super ugly number，找到後更新ptr (有可能同時有兩個最小值，需同時更新ptr)

※此題同時可使用priority queue解。

**870. Advantage Shuffle**

利用greedy algorithm(策略: 對於每個B[i]，找一個A[i]中大於它且最小的一個值，若沒有則配最小的A[i]給它)。使用multiset作為A之容器(sorted and duplicates allowed) ※若使用兩個迴圈掃會超過time limit O(n2) ，所以再找大於B[i]且最小的一個值可利用multiset.upper\_bound() 來找，且time complexity 為 O(nlogn)

2020/08/11

**160. Intersection of Two Linked Lists**

使用兩個pointer從兩個linklist開始遍歷，遍歷完就去遍歷另外一個，到最後若兩個linklist有交集必會同時達到交點(若無交集則同時最終達到null)

2020/12/15

**279. Perfect Squares**

使用 recursion 會超過time limit exceed要使用dp

dp[i + j \* j] = min(dp[i + j \* j], dp[i] + 1);

(其實利用四平方定理，有更好的解法)

2020/12/16

**127. Word Ladder**

將wordlist 放進set中再利用bfs 進行搜索(queue) 。在queue中dequeue搜索，搜索方式為用兩個迴圈，一為改變第幾個字元二為a~z，然後對造是否有在wordset中，有的話則push入queue中然後將此字從set中刪除(不須重複探索)。如果找到endword則回傳沒有則回傳0

**300. Longest Increasing Subsequence**

第一次嘗試用recursion 會超時，要用dp。

O(n2)方法:dp[i]為以ith element為結尾的最大subsequence數量，初始化全部為1。更新方式，以每個i為結尾時掃 1 ~ i – 1(j)，如果nums[j]小於nums[i]則更新dp[i]為max(dp[i], dp[j]+1)。然後再比較更新完dp[i]何者為最大

O(nlogn)方法:

https://medium.com/@bill800227/leetcode-300-longest-increasing-subsequence-168fe36e1e2

我們姑且把這兩個的結尾數用tails[i](粗體數字:2, 3)來表示，其意義為長度為i+1且以tails[i]結尾遞增序列，而今天在加入一個數字num，我們會希望盡量去把tails更新成較小的數字，因為這樣更有可能形成更長的遞增數列，而我們可能有以下三種狀況：

num < tails[0]：這時候我們需要更新長度為1的遞增序列，也就是：tails[0] = num

tails[0] < num < tails [end]：這說明num可以更新一個遞增序列的結果，我們需要利用binary search或者STL裡面的lower\_bound()找到第一個大於或等於num的tail[i]並更新之，這個時間複雜度為O(logn)，也就是：tails[i] = num，where tails[i] ≥ num

num ≥ tails[end]：這說明一個更大長度的遞增序列產生，我們需要產生一個新的tail並且放在最後面，也就是tails.push\_back(num)

2020/12/17

**1143. Longest Common Subsequence**

使用dp解。建一2d矩陣，代表text1 text2 長度為多少時之longest common subsequence。更新方式為若text1[i - 1] == text2[j – 1]時，則dp[i][j] = dp[i – 1][j – 1] + 1；若不相等則dp[i][j]更新為dp[i – 1][j]與dp[i][j – 1]的最大值。(dp[i][0]與dp[0][j]皆為0)

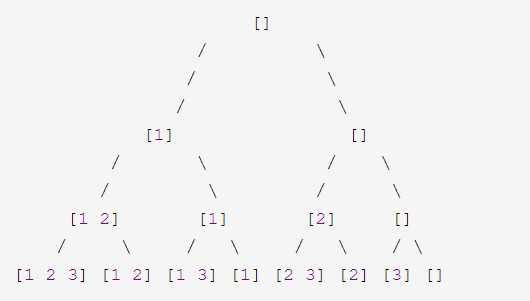
**377. Combination Sum IV**

用dp已解出來，但有幾個testcase會overflow。但網路上茶答案跟我一樣-.-

2020/12/19

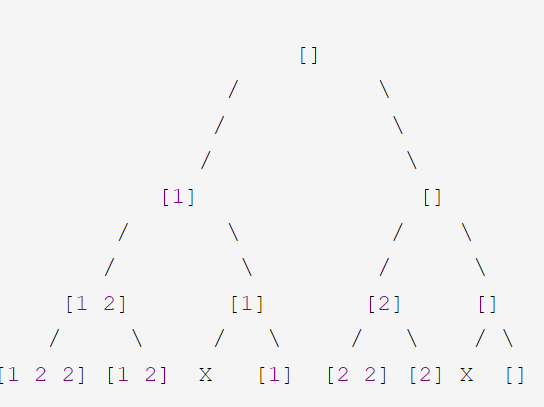
**78. Subsets**

利用recursion，每次可選擇選or不選nums中element。



**90. Subsets II**

調整第78題(Subsets)。在原本第78題，每一個element皆有兩個選項選與不選，在有duplicates時在沒選時直接跳到下一個與現在不同的element，避免duplicate subset。



2020/12/20

**55. Jump Game**

使用greedy解會優於dp。宣告reach為可以到的最遠距離，每次都選擇可走的最大步更新reach。若reach已經比當前位置還小就break(表示達不到這裡)。結束for loop後若reach >= nums.size() – 1則代表可到達。

**1. Two Sum**

不能使用brute force。使用一hash儲存各element位置後在遍歷nums時若發現hash中有target – num[i ]的key則找到。

\*因 unordered\_map之查找為O(1)，所以可以用O(n)解出答案

\*其實不須先建立再查找，可同時(查找若尋無再建立)可將two pass 減為one pass。

\* 網路上的人說只要是two sum的題目第一個就是要想到 hash map。

**15. 3Sum**

將3sum 換減為2sum 問題(先sort然後兩指針往中間找)。然後使用三支指針，第一支指針為fix，就將題目化減為2 sum 然後找0 – num[ptr1]。因為要找出所有triple，所以當找到一組後還ptr2, ptr3還是要繼續往中間移動。\*因為有duplicate，ptr1，ptr2，ptr3都要跳過重複的 (while(nums[ptr]==nums[ptr – 1] ++ptr)。

2020/12/21

用dp解。

用兩個array local global。Local[i][j]表示第i天總共交易j次且最後一次交易在最後一天最大收益，Global[i][j] 表示第i天總共交易j次最大收益。更新遞迴:

**local[i][j] = max(global[i - 1][j - 1], local[i - 1][j]) + diff;**

**global[i][j] = max(global[i - 1][j], local[i][j]);**

※local更新：

最後一張前一天買的話global[i – 1][j – 1] + diff

更早之前買最後一張的話local[i – 1][j] + diff

兩個比較最大值更新local[i][j]。

※global更新：

最後一隻最後一天賣出local[i][j]

最後一隻不是最後一天global[i – 1][j]

兩個更新global[i][j]。

2020/12/23

**33. Search in Rotated Sorted Array**

一樣用二分法，但使用前先用mid與r比較大小，若r>mid則代表右邊有序；r<mid則代表左邊有序，則比較target是否落在有序區間內，有保留無捨棄這半邊。\*(這題若用mid與l比較大小，則在元素剩下兩個的時候有誤，因為這時mid會 == l，因此會判斷錯誤)

**11. Container With Most Water**

用dp o(n^2)會超時，用兩個l跟r指向兩端往中間，移動l 跟 r中較小的那個，並查開新的pointer為出來的區域，若比最大值大則更新。

2020/12/24

**417. Pacific Atlantic Water Flow**

原本從高點往海洋找結果陷入無限迴圈(無法紀錄有沒有到達過，且這樣搜索展開太多)。看詳解解題思路：從四邊的海開始往上找，若可以繼續往上則代表可以到達。建立二尺寸跟matrix一樣的matrix來記錄是否可以從pacific到達和atlantic到達，然後由四個邊使用dfs找查。最後再用兩個迴圈同時找查reach Pacific 跟reach Atlantic，若兩個同時為true則將此結果加入結果。

**73. Set Matrix Zeroes**

原解法：用兩vector row跟column來記錄該行該列是否為0，然後再掃一次矩陣，若column[j]或row[i]為0，則將matrix[i][j]設為0

Time: O(mn) Space: O(m + n)

網上最佳解：用matrix的第 0 row跟第0 column來取代原本創建的row 和 column。並設兩個flag column0跟 row 0表示原本第零列第零行是否為0。

- 先扫描第零行第零列，如果有0，则将各自的flag设置为true  
- 然后扫描除去第零行第零列的整个数组，如果有0，则将对应的第零行和第零列的数字赋0  
- 再次遍历除去第零行第零列的整个数组，如果对应的第零行和第零列的数字有一个为0，则将当前值赋0  
- 最后根据第零行第零列的flag来更新第零行第零列

**54. Spiral Matrix**

硬尻。畫圖發現走直跟走恆都是上次的減一，當減到0時代表結束(畫圖解)

2020/12/26

**48. Rotate Image**

**79. Word Search**

使用dfs解，在dfs時不能走重複的路，故dfs時要把探索的點label visited，但探索結束後要重新label為unvisited，讓下一次新的探索可以重找

**50. Pow(x, n)**

用recusion解pow(x,n) => pow(x, n/2) \* pow(x, n/2)的概念。然後用 n%2 == 0? 來決定要步要在 \* x(或/x n<0的話)

2020/12/27

**57. Insert Interval**

網解：如果newInterval[0]跟[1]都大於interval，則直接加入，如果newInterval已經比較小了，則不停地比較intervals跟newInterval[1]直接取代他們直到下一個interval[0]比newInterval[1]還大，然後就讓剩下的intervals也加入。

**56. Merge Intervals**

與57提類似，先sort(第一個職越小放越前面)。先把第一個放入ret，然後利用一個ptr遍歷一次解決。如果intervals[ptr][0]比上一個放入的第二個元素(end interval) 大，則直接push\_back，沒有的話則直接修改ret.back()的end interval直

2020/12/28

**23. Merge k Sorted Lists**

利用priority queue。先將所有node放入priority queue中，然後再不斷地把queue.top取出，成為merge list。 Easy peasy using priority queue

**347. Top K Frequent Elements**

先用unordered\_map紀錄每個元素出現的次數，然後再把map中element與次數以pair的方式放入priority queue，之後再取出。

亦可用bucket sort

2020/12/29

**141. Linked List Cycle**

用快慢指針，若快指針走到底則回傳false，若快指針跟曼指針相等則true(快指針不可能走到底)。

**143. Reorder List**

三步驟：

1. 找出中間斷點
2. 將後面那段翻轉
3. 依序放入新融合list

※翻轉可使用stack

2020/12/30

**227. Basic Calculator II**

使用一個stack來存數字並用op來存現在的運算符號，現在運算符號op左邊的數字已經放入stack中(所以while loop前先push(0)，然後再用while loop 的pointer遍歷。若下一個是符號是’+’，則把op右邊的數加入stack;若下一個是’-‘則把現在數字取負號放入stack；若下一個是’\*’則把現在這個數字乘與stack.top (op左邊的數字) 然後再重新放入stack中，若op是’/’處理跟’\*’類似。

最後再把stack中所有東西加起來就是答案

2021/01/03-04

**424. Longest Repeating Character Replacement**

使用sliding windows方式找查。若窗戶長度 – 最常見元素數量 <= k 代表此窗戶之子字串可以變為連續，所以可以增大窗戶(ptr\_r ++)。此題做一變形，若不符合(無法在k步操作內用回)也不須減小窗戶，僅需滑動，因接下來我們只在意是否有更大於原本窗戶的結果。

**49. Group Anagrams**

因為anagrams為每個字母出現的次數相同，就為同種anagrams，因此只要sort後兩個字為相同及為anagram。因此遍歷str，以sort後的字為key，用一unordered\_map<string, vector<string>>紀錄，若map中沒有此key則增加，有此key則在map的value push\_back此string。

**5. Longest Palindromic Substring**

有兩方法

Dp:

Dp[i][j]代表由j ~ i 之substring 是否為回文。更新遞迴方式：

若I == j 則dp[i][j] = true

若 I == j + 1 ，若 s[i] == s[j]則 dp[i][j]為回文

若其他 則dp[i][j]= (dp[I – 1][j+ 1] && s[i] == s[j]) (比較頭尾兩字，然後找第二個字與倒數第二個字的dp)

如果dp[i][j]為真，則比較此substring與最長長度，若更長則更新長度與起點

用兩個迴圈更新，第一個迴圈為substring結尾在哪，第二個迴圈為開頭在哪

Expand from middle:

遍歷依次string並輪流把element當作回文的中點，並往外延伸回文可能長度(若左右兩字相等，則可再增加左右兩ptr)。一個點分別有兩種回文可能，分別為基數型回文與偶數型回文，基數型回文就以遍歷到的這點當起點往外延伸，偶數型則以遍歷到的點加與其後面一點當起點開始延伸。

**516. Longest Palindromic Subsequence**

與longest palindromic substring 類似，但差別時若發現不同時substring就須直接停止，但subsequence需要繼續找。故使用dp解，解法類似substring，Dp:

Dp[i][j]代表由j ~ i 之substring 是否為回文。更新遞迴方式：

若I == j 則dp[i][j] = 1

若 I == j + 1 ，若 s[i] == s[j]則 dp[i][j]為2不相等則為1

若其他

若s[i] == s[j]則dp[i][j] = dp[i – 1][j + 1] + 2 (最外面相等直接比較裡面一層)

若不相等則dp[i][j]=max(dp[i-1][j],dp[i][j+1]) (因為不相等，可以看過最尾or 最頭，比較取最大的)

便利完後找dp[s.size()-1][0]極為答案

**155. Min Stack**

使用兩個stack，一個當作正常stack，push pop皆跟一般stack依樣；一個用於存放最小值，在push時若minstack中沒有值或是minstack.top比較大則push入，成為新的Min，在pop時若minstack.top跟realstack.top是一樣的話則一起pop出，表示被pop的元素剛好為最小值。

2021/01/05

**162. Find Peak Element**

利用 binary search，若mid比兩邊還大就找到，若右邊最大則l=mid + 1若左邊最大則r =mid – 1

**1344. Angle Between Hands of a Clock**

先找出時針與分針角度，找到後大減小，如果小於180極為答案大於180則360-此樹為答案

2021/01/08

**1662. Check If Two String Arrays are Equivalent**

使用方式：用四個pointer，兩個分別只word1的一維與二維，另二個只word2的一維與二維，然後一個一個比較到底(爛-.-)

Better solution:把所有word1 與 word2的所有element年在一起，再看看string1 與 string2是否相同