Lyft Leetcode

Given an N digit number, find all possible numbers that can be composed from it without reusing a digit. For example: 123 -> (1, 2,3, 12, 13, 21, 23, 31, 32, 123, 132, 213, 231, 312, 321)

```
def permute(nums):
    permutations = [[]]
    for num in str(nums):
        permutations.extend([p[:j] + [num] + p[j:] for p in permutations for j in range(len(p)+1)])
    result = []
    for num in permutations:
        if num:
            result.append(int("".join(num)))
    return result
```

443. String Compression
aaabbbbccc
壓縮成3a4b3c

```
def compress(chars):
    currInd = 0
    last = 0
    chars = [c for c in chars]

    while currInd < len(chars):
        current_char = chars[currInd]
        lookAhead = currInd + 1
        count = 1
        while lookAhead < len(chars) and chars[lookAhead] == chars[currInd]:
            count += 1
            lookAhead += 1
        if count > 1:
            for d in str(count):
                chars[last] = d
                last += 1
        chars[last] = current_char
        last += 1
        currInd = lookAhead
    return "".join(chars[:last])
```

7. Reverse Integer

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

```
    def reverse(self, x):
            if not self.isValid(x):
                return 0
            result = 0
            sign = -1 if x < 0 else 1
            x = abs(x)
            while x > 0:
                result, x = result*10 + x % 10, x //10
            result = sign * result
            return result if self.isValid(result) else 0

        def isValid(self, x):
            return -2**31 <= x <= 2**31 -1
```

Reverse an int x (turn it to pos first):

result, x = result*10 + x % 10, x //10

---

42. Trapping Rain Water
Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

The above elevation map is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped. Thanks Marcos for contributing this image!

Example:

Input: [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6

---

44. Wildcard Matching
Given an input string (s) and a pattern (p), implement wildcard pattern matching with support for '?' and '*'.

'?' Matches any single character.
'*' Matches any sequence of characters (including the empty sequence).
The matching should cover the entire input string (not partial).
2
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 *.
Example 1:

Input:
s = "aa"
p = "a"
Output: false
Explanation: "a" does not match the entire string "aa".
Example 2:

Input:
s = "aa"
p = "*"
Output: true
Explanation: '*' matches any sequence.

---

69. Sqrt(x)
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.

---

155. Min Stack
DescriptionHintsSubmissionsDiscussSolution
Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

push(x) -- Push element x onto stack.
pop() -- Removes the element on top of the stack.
top() -- Get the top element.
getMin() -- Retrieve the minimum element **in the stack. //we don't care those popped away**
Example:
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin();   --> Returns -3.
minStack.pop();
minStack.top();   --> Returns 0.
minStack.getMin();   --> Returns -2.

```
class MinStack:

    def __init__(self):
        self.minstack = []
        self.stack = []

    def push(self, x):
        self.stack.append(x)
        if not self.minstack or x <= self.minstack[-1]:
            self.minstack.append(x)

    def pop(self):
        if self.stack.pop() == self.minstack[-1]:
            self.minstack.pop()


    def top(self):
        return self.stack[-1]

    def getMin(self):
        return self.minstack[-1]
```

---

157. Read N Characters Given Read4
The API: int read4(char *buf) reads 4 characters at a time from a file.

The return value is the actual number of characters read. For example, it returns 3 if there is only 3 characters left in the file.

By using the read4 API, implement the function int read(char *buf, int n) that reads n characters from the file.

Example 1:

Input: buf = "abc", n = 4
Output: "abc"
Explanation: The actual number of characters read is 3, which is "abc".
Example 2:

Input: buf = "abcde", n = 5
Output: "abcde"

```
    def read(self, buf, n):
        i = 0
        while i < n:
            buf4 = [""]*4
            count = min(n-i, read4(buf4))
            if not count:
                return i
            for j in range(count):
                buf[i] = buf4[j]
                i += 1
        return i
```

In fact, the read() method just does two things:

> It returns an int to show how long the file is.
> After we pass char[] to the read() method, and the char[] is filled with characters extracted from the files by read().
> We fill the buf with those characters.

Read N Characters Given Read4 II - Call multiple times

```
    def __init__(self):
        from collections import deque
        self.q = deque()
    def read(self, buf, n):
        i = 0
        while True:
            buf4 = [""]*4
            count = read4(buf4)
            self.q.extend(buf4)
```

```
            cur = min(len(self.q), n-i)
            if not cur:
                return i
            for _ in range(cur):
                buf[i] = self.q.popleft()
                i += 1
        return i
```

Call once: Assume you are always going to read from the start of the file/buffer.
Call multiple times: Start reading from where you left off. This means that you have to store the last place (ptr) where you stopped and store the read but uncopied bytes to the buffer.

---

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

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

Example 1:

Input: [3,2,3]
Output: 3
Example 2:

Input: [2,2,1,1,1,2,2]
Output: 2

```
    def majorityElement(self, nums):
        if not nums:
            return None
        cand, count = nums[0], 0
        for i, cur in enumerate(nums):
            if count == 0:
                cand = cur
                count = 1
            elif cur == cand:
                count += 1
            else:
                count -= 1
        return cand
```

Majority voting

---

200. Number of Islands
Given a 2d grid map of '1's (land) and '0's (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

Input:
11110
11010
11000
00000

Output: 1
Example 2:

Input:
11000
11000
00100
00011

Output: 3

对角线也算同一岛
讨论 DFS, BFS 复杂度

```
    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(i, j, grid) 或者 self.bfs(i, j, grid)
                    count += 1
        return count


     def dfs(self, i, j, grid):
        grid[i][j] = "-1"
        for x, y in [(i+1, j), (i-1, j), (i, j-1), (i, j+1)]:
            if 0 <= x < len(grid) and 0 <= y < len(grid[0]) and grid[x][y] == "1":
                self.dfs(x, y, grid)
        return

    def bfs(self, i, j, grid):
        grid[i][j] = "0"
        d = deque([(i, j)])
        while d:
            i, j = d.popleft()
            for x, y in [(i+1, j), (i-1, j), (i, j-1), (i, j+1)]:
                if 0 <= x < len(grid) and 0 <= y < len(grid[0]) and grid[x][y] == "1":
                    d.append((x, y))
                    grid[x][y] = "0"


    Union Find
     def find(self, parent, a):
        if parent[a] != a:
            parent[a] = self.find(parent, parent[a])
```

```
            return parent[a]

        def union(self, size, parent, a, b, count):
            a, b = self.find(parent, a), self.find(parent, b)
            if a != b:
                if size[a] < size[b]:
                    a, b = b, a
                size[a] += size[b]
                parent[b] = a
                count[0] -= 1


        def numIslands(self, grid):
            """
            :type grid: List[List[int]]
            :rtype: int
            """
            count = [0]
            parent, size = {}, {}
            for i in range(len(grid)):
                for j in range(len(grid[0])):
                    if grid[i][j] == "1":
                        parent[i, j] =  (i, j)
                        size[i, j] = 1
                        for x, y in [(i+1, j), (i-1, j), (i, j+1), (i, j-1)]:
                            if 0 <= x < len(grid) and 0 <= y < len(grid[0]) and (x, y) in
    parent:
                                self.union(size, parent, (i, j), (x, y), count)
                        count[0] += 1
            return count[0]
```

For all three methods:
Time complexity : O(mn) because we traverse each cell once, regardless of whether it's 1 or 0

Space complexity : worst case O(mn) in case that the grid map is filled with lands where DFS goes by M×N deep.

---

215. Kth Largest Element in an Array
Find the kth largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

Example 1:

Input: [3,2,1,5,6,4] and k = 2
Output: 5
Example 2:

Input: [3,2,3,1,2,4,5,5,6] and k = 4
Output: 4
Sol 1: Quick Select O(n)

```
```

```
    def findKthLargest(self, nums, k):
            smaller, bigger = [], []
            pivot = random.choice(nums)
            for n in nums:
                if n < pivot:
                    smaller.append(n)
                elif n > pivot:
                    bigger.append(n)
            if k <= len(bigger):
                return self.findKthLargest(bigger, k)
                #k > len(bigger) + 1 is incorrect
            if k > len(nums)-len(smaller):
                return self.findKthLargest(smaller, k-(len(nums) - len(smaller)))
            return pivot
```

Divide and Conquer
Note that **len(smaller) + len(bigger) + 1 != len(nums)** because there might be duplicates.

Sol 2 Heap O(k*logn), Space:O(k)

```
    def findKthLargest(self, nums, k):
            heap = nums[:k]
            heapify(heap)
            for n in nums[k:]:
                if n > heap[0]:
                    heappushpop(heap, n)
            return heap[0]
```

---

230. Kth Smallest Element in a BST
Given a binary search tree, write a function kthSmallest to find the kth smallest element in it.

Note:
You may assume k is always valid, 1 ≤ k ≤ BST's total elements.

Example 1:

Input: root = [3,1,4,null,2], k = 1
```
   3
  / \
 1   4
  \
   2
```
Output: 1
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? **Use generator**
1. Iterative

```
    def kthSmallest(self, root, k):
```

```
        if not root:
            return None
        stack = []
        while stack or root:
            while root:
                stack.append(root)
                root = root.left
            root = stack.pop()
            if k == 1:
                return root.val
            root = root.right
            k -= 1
```

2. Generator

```
    def kthSmallest(self, root, k):
        for val in self.inOrder(root):
            if k == 1:
                return val
            else:
                k -= 1

    def inOrder(self, root):
        if not root:
            return
        for val in self.inOrder(root.left):
            yield val

        yield root.val

        for val in self.inOrder(root.right):
            yield val
```

3. Recursion

```
    def kthSmallest(self, root, k):
        self.k = k
        self.res = None
        self.inOrder(root)
        return self.res

    def inOrder(self, root):
        if not root:
            return
        self.inOrder(root.left)
        self.k -= 1
        if self.k == 0:
            self.res = root.val
            return
        self.inOrder(root.right)
```

Union Find

305. Number of Islands II
A 2d grid map of m rows and n columns is initially filled with water. We may perform an addLand operation which turns the water at position (row, col) into a land. Given a list of positions to operate, count the number of islands after each addLand operation. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example:

Input: m = 3, n = 3, positions = [[0,0], [0,1], [1,2], [2,1]]
Output: [1,1,2,3]
Explanation:

Initially, the 2d grid grid is filled with water. (Assume 0 represents water and 1 represents land).

0 0 0
0 0 0
0 0 0
Operation #1: addLand(0, 0) turns the water at grid[0][0] into a land.

1 0 0
0 0 0 Number of islands = 1
0 0 0
Operation #2: addLand(0, 1) turns the water at grid[0][1] into a land.

1 1 0
0 0 0 Number of islands = 1
0 0 0
Operation #3: addLand(1, 2) turns the water at grid[1][2] into a land.

1 1 0
0 0 1 Number of islands = 2
0 0 0
Operation #4: addLand(2, 1) turns the water at grid[2][1] into a land.

1 1 0
0 0 1 Number of islands = 3
0 1 0
Follow up:

Can you do it in time complexity O(k log mn), where k is the length of the positions?

```
    def union(self, size, parent, a, b, count):
        a, b = self.find(parent, a), self.find(parent, b)
        if a != b:
            if size[a] < size[b]:
                a, b = b, a
```

```
                parent[b] = a
                size[a] += size[b]
                count[0] -= 1

        def find(self, parent, a):
            if parent[a] != a:
                parent[a] = self.find(parent,parent[a])
            return parent[a]


        def numIslands2(self, m, n, positions):
            parent, size = {}, {}
            count = [0]
            res = []
            for i, j in positions:
                parent[i, j] = (i, j)
                size[i, j] = 1
                for x, y in [(i+1, j), (i-1, j), (i, j-1), (i, j+1)]:
                    if 0 <= x < m and 0 <= y <n and (x, y) in parent:
                        self.union(size, parent, (i, j), (x, y), count)
                count[0] += 1
                res.append(count[0])
            return res
```

Union by size
Initializing an instance of disjoint sets: O(1) (dict works better in case of a huge grid, with a small amount of islands.)
Performing Union() in any of the implementations: O(1)
Performing Find() in Union by size or height: O(log(n))

Union by rank
always attaches the shorter tree to the root of the taller tree. Thus, the resulting tree is no taller than the originals unless they were of equal height, in which case the resulting tree is taller by one node.

To implement union by rank, each element is associated with a rank. Initially a set has one element and a rank of zero. If two sets are unioned and have the same rank, the resulting set's rank is one larger; otherwise, if two sets are unioned and have different ranks, the resulting set's rank is the larger of the two. Ranks are used instead of height or depth because path compression will change the trees' heights over time.

---

279. Perfect Squares
Given a positive integer n, find the least number of perfect square numbers (for example, 1, 4, 9, 16, ...) which sum to n.

Example 1:

Input: n = 12
Output: 3
Explanation: 12 = 4 + 4 + 4.
Example 2:

Input: n = 13
Output: 2
Explanation: 13 = 4 + 9.

---

443. String Compression
Given an array of characters, compress it in-place.

The length after compression must always be smaller than or equal to the original array.

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 array.

Follow up:
Could you solve it using only O(1) extra space?

Example 1:
Input:
["a","a","b","b","c","c","c"]

Output:
Return 6, and the first 6 characters of the input array should be: ["a","2","b","2","c","3"]

Explanation:
"aa" is replaced by "a2". "bb" is replaced by "b2". "ccc" is replaced by "c3".

```
def compress(self, chars):
        i = j = 0
        while j < len(chars):
            num = 1
            while j + 1 < len(chars) and chars[j] == chars[j+1]:
                num += 1
                j += 1
            if 1 < num < 10:
                chars[i] = chars[j-num+1]
                chars[i+1] = str(num)
                i += 2
            elif num == 1:
                chars[i] = chars[j]
                i += 1
            else:
                chars[i] = chars[j-num+1]
                for k in str(num):
                    i += 1
                    chars[i] = k
                i += 1
            j += 1
        return i
```

501. Find Mode in Binary Search Tree
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],

```
  1
   \
    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).

```
    def findMode(self, root):
        return self.inOrder(1, 0, None, root, [])[3]

    def inOrder(self, cnt, maxCnt, prev, root, result):
        if not root:
            return cnt, maxCnt, prev, result
        cnt, maxCnt, prev, result = self.inOrder(cnt, maxCnt, prev, root.left, result)
        if prev:
            if prev.val == root.val:
                cnt += 1
            else:
                cnt = 1
        if cnt > maxCnt:
            maxCnt = cnt
            del result[:]
            result.append(root.val)
        elif cnt == maxCnt:
            result.append(root.val)
        return self.inOrder(cnt, maxCnt, root, root.right, result)
```

vanilla:

```
    def findMode(self, root):
```

```
        if not root:
            return []
        counter = collections.Counter()
        self.inOrder(root, counter)
        max_val = max(counter.values())
        return [k for k, v in counter.items() if v == max_val]
    def inOrder(self, root, counter):
        if root:
            counter[root.val] +=1
            self.inOrder(root.left, counter)
            self.inOrder(root.right, counter)
```

735. Asteroid Collision
We are given an array asteroids of integers representing asteroids in a row.

For each asteroid, the absolute value represents its size, and the sign represents its direction (positive meaning right, negative meaning left). Each asteroid moves at the same speed.

Find out the state of the asteroids after all collisions. If two asteroids meet, the smaller one will explode. If both are the same size, both will explode. Two asteroids moving in the same direction will never meet.

Example 1:
Input:
asteroids = [5, 10, -5]
Output: [5, 10]
Explanation:
The 10 and -5 collide resulting in 10. The 5 and 10 never collide.

Input:
asteroids = [-2, -1, 1, 2]
Output: [-2, -1, 1, 2]
Explanation:
The -2 and -1 are moving left, while the 1 and 2 are moving right.
Asteroids moving the same direction never meet, so no asteroids will meet each other.
Note:

The length of asteroids will be at most 10000.
Each asteroid will be a non-zero integer in the range [-1000, 1000]

```
    def asteroidCollision(self, asteroids):
        stack = []
        for a in asteroids:
            if a > 0:
                stack.append(a)
            elif not stack or stack[-1] < 0:
                stack.append(a)
            else:
```

```
                append = True
                while stack and 0 < stack[-1] <= abs(a):
                    if stack[-1] == abs(a):
                        append = False
                        stack.pop()
                        break
                    stack.pop()


                if not stack and append or stack[-1] < 0 and append:
                    stack.append(a)
        return stack
```

Constant space sol !!

```
def asteroidCollision(self, asteroids):
       #asteroid going right
        r = 0
        #asteroid going left
        l = 1
        while l < len(asteroids):
            if r == -1:
                r = 0
                asteroids[r] = asteroids[l]
                l += 1
            elif asteroids[r] > 0 and asteroids[l] < 0:
                if asteroids[r] == abs(asteroids[l]):
                    r -= 1
                    l += 1
                elif asteroids[r] < abs(asteroids[l]):
                    r -= 1
                else:
                    l += 1
            else:
                r += 1
                asteroids[r] = asteroids[l]
                l += 1
        return asteroids[:r+1]
```

Buy and Sell stock

```
def maxProfit(self, prices):
    """
    :type prices: List[int]
    :rtype: int
    """
    min_price = float("inf")
    max_profit = 0
    for price in prices:
        max_profit = max(max_profit, price - min_price)
        min_price = min(min_price, price)
```

```
        return max_profit
```

12号的第一轮，白人小哥，非常nice，题目是他家常考的driver mode题。大意是，利福特需要检查司机是否被允许进入drive mode，允许的条件是，司机在上一次连续八小时休息之后，驾驶时间不超过12个小时。input是一系列的shift，shift是类似[3, 12], [19, 26]这样的数组，表示司机驾驶的开始时间和结束时间。比如[3, 12]就是表示司机从时刻3开始驾驶，到时刻12结束。要求判断司机是否可以进入driver mode。
follow up是给定起止时间，如何计算这段时间之内司机总共的驾驶时间。
function to determine whether the driver is allowed to enter driver mode
* drivers are not allowed to drive a total of 12 hours without an 8 hour
break
* the function inputs are:
- a list of driver shifts as start/end integers, the integer is relative to
lyft launch
- the current time since lyft launch as integer

def can_drive(history, current_time):
    """
    Returns true if the driver has driven less than 12 hours since their
last 8 hour break

    history|array - Shifts, e.g. [(0, 12), (13, 19)]
    current_time|int - Current timestamp as hour since Lyft launch, e.g. 50

    can_drive = True example
        # 9 hour break, 1 hour shift, 2 hour break, 10 hour shift
        history = [(9, 10), (12, 22)]
        current_time = 24

    can_drive = False example:
        history = [(0, 4), (5, 9), (10, 14), (15, 19), (20, 24)]
        current_time = 24

```
def can_drive(history, current_time):
    history.append((current_time, current_time))
    work = rest = last = 0
    for start, end in history[::-1]:
        work += end - start
        if last - end >= 8:
            return True
        if work >= 12:
            return False
        last = start
    return True
```

> follow up是给定起止时间，如何计算这段时间之内司机总共的驾驶时间。” 是什么意思?

就是假如给定的shift是[2, 10], [13, 15], [18, 28]，给定起止时间3， 20，求这段时间内司机总共驾驶了多久，这里就是11. 我当时说二分，他认可了。

```
from bisect import *

def total_worktime(history, start, end):
    #bisect algorith is binary search, log(n)
    start_ind = bisect_right(history, [start, start])
    end_ind = bisect_right(history, [end, end])

    #case 1 & 2: start and end are 1.earlier than the first shift 2.later than the last shift
    if start_ind == end_ind == 0 or start_ind == len(history):
        return 0
    #case 3: start and end are in between two shifts
    elif start_ind == end_ind and history[start_ind][1] <= start:
        return 0
    #case 4:
    else:
        work = 0
        # start and end are within the previous shift
        if start_ind > 0:
            prev_end = history[start_ind-1][1]
            if start < prev_end:
                work += min(prev_end, end) - start
        # add the shifts in between start and end
        while start_ind < end_ind:
            prev_end, prev_start = history[start_ind][1], history[start_ind][0]
            work += min(prev_end, end) - prev_start
            start_ind += 1
        return work
```

19号的第二轮是印度大哥，人也很和蔼。题目是利特口盛雨水题原题。写完之后跑了几个test case大哥忽然跟我说，你试试输入数组包含0的test case，我就试了一下，然后大哥说结果不对，我一脸懵逼，结果他说高度为0 的地方是个洞。。。雨水会全部漏掉。。。然后问我怎么处理，我就说在原来的while loop里加个判断当前高度是否为0，时间不够了就没写code。他之前只说传入的是一个代表高度的数组完全没提有洞的事，所以当时真的是一脸懵。。。

```
#0 = hole
def trap(height):
    n = len(height)
    l, r, water, minHeight = 0, n - 1, 0, 0
    block = 0
    while l < r:
        minHeight = min(height[l], height[r])
        while l < r and height[l] <= minHeight:
            if height[l] == 0:
                block = 0
                l += 1
                while l < r and height[l] < minHeight:
```

```
                    l += 1
            else:
                block += minHeight - height[l]
                if height[l] == minHeight:
                    water += block
                    block = 0
                l += 1
        water += block
        block = 0
        while r > l and height[r] <= minHeight:
            if height[r] == 0:
                block = 0
                r -= 1
                while l < r and height[r] < minHeight:
                    r -= 1
                print(height[r],r,minHeight)
            else:
                block += minHeight - height[r]
                if height[r] == minHeight:
                    water += block
                    block = 0
                r -= 1
        water += block
        block = 0
    return water
#Regular
#1. Stack, O(n) time&space

#2. Two pointers


```

Print all interleavings of given two strings

```
#Recursive
def interleave(a, b):
    if len(a) == 0 and len(b) == 0:
        return []
    if len(a) == 0:
        return [b]
    if len(b) == 0:
        return [a]
    arr1 = [a[0] + s for s in interleave(a[1:], b)]
    arr2 = [b[0] + s for s in interleave(a, b[1:])]

    return arr1 + arr2
```

If the strings are length m and n and printing takes constant time, the whole job can be done in $O(\binom{m+n}{n})$ because there are $\binom{m+n}{n}$ different interleavings in the case that all characters in the strings are distinct.

```
#Iterative # i don't understand
def interleave(a, b):
    result = [[]]
    for i in range(1, len(b) + 1):
        result.append([b[0:i]])


    for i in range(len(a)):
        result[0] = [a[0:i + 1]]

        for j in range(1, len(b) + 1):
            print( result[j], result[j-1])
            first = []
            for s in result[j]:
                first.append(s + a[i])

            second = []
            for s in result[j - 1]:
                second.append(s + b[j - 1])
            result[j] = first + second

    return result[len(b)]
```

把两个dict合并成一个，每个dict key的value可能是另一个dict，或才是int，如果是key相同，value是int，就相加，如果value是dict就再合并。如果你们有兴趣我可以具体写
然后我就指出了他题目里的一个问题，如果 dictA['a'] 是 int，dictB['a'] 是 dict 怎么办，他说不会这样。那好，简单的递归，十分钟写完，跑了test case没问题，其中有一个小bug，跑完第一次之后改过来。

```
def merge_dic(dicA, dicB):
    from collections import defaultdict
    result = defaultdict(int)
    helper(result, dicA)
    helper(result, dicB)
    return result
def helper(result, dic):
    for key, value in dic.items():
        if isinstance(value, int):
            result[key] += value
        else:
            helper(result, value)
#not flattening dictionary
def merge_dic(dicA, dicB):
    result = {}
    for key in dicA:
        print(key)
```

```
        if isinstance(dicA[key], int):
            result[key] = result.get(key, 0) + dicB[key] + dicA[key]
        else:
            result[key] = merge_dic(dicA[key],dicB[key])
    return result
```

Given an N digit number, find all possible numbers that can be composed from it without reusing a digit. For example: 123 -> (1, 2,3, 12, 13, 21, 23, 31, 32, 123, 132, 213, 231, 312, 321)

269.Alien Dictionary

```
    #Topological sort, O(len(word)+#unique order)
    #                  O(V + E)

    def alienOrder(self, words):
        """
        :type words: List[str]
        :rtype: str
        """
        from collections import defaultdict, deque

        chars = set("".join(words))
        indegrees = {char : 0 for char in chars}
        suc = defaultdict(list)
        for i in range(len(words)-1):
            for a, b in zip(words[i], words[i+1]):
                if a != b:
                    indegrees[b] += 1
                    suc[a].append(b)
                    break

        queue = deque(filter(lambda x: indegrees[x] == 0, indegrees.keys()))
        order = ""
        while queue:
            x = queue.popleft()
            order += x
            for c in suc[x]:
                indegrees[c] -= 1
                if indegrees[c] == 0:
                    queue.append(c)

        return order * (set(order)==chars)
```

Lowest Common Ancestor of a Binary Tree

```
#Easy recursion
 def lowestCommonAncestor(self, root, p, q):
        if not root:
            return None
        if p is root or q is root:
            return root
```

```
        left = self.lowestCommonAncestor(root.left, p, q)
        right = self.lowestCommonAncestor(root.right, p, q)
        if left and right:
            return root
        return left or right
#Follow-up
#  (1) 如果输入的node不在树里面怎么办
#  (2) 如果两个node是相同的怎么办
#  (2) 如果有两个Thread同时访问这棵树会不会有问题? 可不可以有办法不用synchronized?

def helper(root, p, q, res):
  if not root:
    return 0

  left = helper(root.left, p, q, res)
  right = helper(root.right, p, q, res)
  found = 0

  if left == 1 and right == 1:
    res[0] = root
  if root == p or root == q:
    found += 1
    res[0] = root
  if root == p == q:
    found += 1
  return found + left + right


def lowestCommonAncestor(root, p, q):
  res = [0]
  if helper(root, p, q,res) == 2:
    return res[0]
  return None
```

63. Unique Paths with Obstacles

```
   # Num of paths at any given pt only depends on up and left cell, thus, if up and left are both obstacles,the cell will be 0
   def uniquePathsWithObstacles(self, obstacleGrid):
        if not obstacleGrid:
            return 0
        row, col = len(obstacleGrid), len(obstacleGrid[0])
        path = [[0]*col for _ in range(row)]
        for i in range(row):
            for j in range(col):
                if obstacleGrid[i][j]:
                    continue
                if i == j == 0:
                    path[i][j] = 1
                else:
                    path[i][j] = path[i-1][j] + path[i][j-1]
```

```
        return path[row-1][col-1]
```

```
import heapq
class MaxStack:

    def __init__(self):
        self.stack = []
        self.heap = []
        self.heapID = set() # id of items deleted in stack but not hp
        self.stackID = set() # id of items deleted in hp but not stack
        self.id = 0

    def push(self, x):
        self.stack.append((x, self.id))
        heapq.heappush(self.heap, (-x, -self.id))
        self.id += 1


    def pop(self):
        x = self.top()
        self.heapID.add(self.stack[-1][1])
        self.stack.pop()
        return x


    def top(self):
        while self.stack and self.stack[-1][1] in self.stackID:
            self.stackID.remove(self.stack[-1][1])
            self.stack.pop()
        return self.stack[-1][0]

    def peekMax(self):
        while self.heap and -self.heap[0][1] in self.heapID:
            self.heapID.remove(-self.heap[0][1])
            heapq.heappop(self.heap)
        if self.heap:
            return -self.heap[0][0]
        return None


    def popMax(self):
        x = self.peekMax()
        self.stackID.add(-self.heap[0][1])
        heapq.heappop(self.heap)
        return x
```