Lyft leetcode

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.
Example 2:
Input:
asteroids = [8, -8]
Output: []
Explanation:
The 8 and -8 collide exploding each other.
Example 3:
Input:
asteroids = [10, 2, -5]
Output: [10]
Explanation:
The 2 and -5 collide resulting in -5. The 10 and -5 collide resulting in 10.
Example 4:
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]..

```python
class Solution:

    def asteroidCollision(self, asteroids):
        """
        :type asteroids: List[int]
        :rtype: List[int]
        """
        after_col = []
        for a in asteroids:
            while after_col and a < 0 < after_col[-1]:
                if -a > after_col[-1]:
                    after_col.pop()
                    continue
                elif -a == after_col[-1]:
                    after_col.pop()
                break
            else:
                after_col.append(a)
        return after_col
```

238. Product of Array Except Self

Given an array nums of n integers where n > 1, return an array output such that output[i] is equal to the product of all the elements of nums except nums[i].

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 as extra space for the purpose of space complexity analysis.)

```python
class Solution:
    def productExceptSelf(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]
        """
        p = 1
        n = len(nums)
        output = []
        for i in range(n):
            output.append(p)
            p *= nums[i]
        p = 1
```

```
        for i in range(n - 1, -1, -1):
            output[i] *= p
            p *= nums[i]
        return output
```

716. Max Stack

Design a max stack that supports push, pop, top, peekMax and popMax.

push(x) -- Push element x onto stack.
pop() -- Remove the element on top of the stack and return it.
top() -- Get the element on the top.
peekMax() -- Retrieve the maximum element in the stack.
popMax() -- Retrieve the maximum element in the stack, and remove it. If you find more than one maximum elements, only remove the top-most one.
Example 1:
MaxStack stack = new MaxStack();
stack.push(5);
stack.push(1);
stack.push(5);
stack.top(); -> 5
stack.popMax(); -> 5
stack.top(); -> 1
stack.peekMax(); -> 5
stack.pop(); -> 1
stack.top(); -> 5
Note:
-1e7 <= x <= 1e7
Number of operations won't exceed 10000.
The last four operations won't be called when stack is empty.

```
from heapq import *
class MaxStack:

    def __init__(self):
        """
        initialize your data structure here.
        """
        self.ls = []
        self.hp = []
        self.lsd = set()
        self.hpd = set() # id of items deleted in ls but not hp
        self.id = 0

    def push(self, x):
        """
        :type x: int
```

```
        :rtype: void
        """
        self.ls.append((x, self.id))
        heappush(self.hp, (-x, -self.id))
        self.id += 1

    def pop(self):
        """
        :rtype: int
        """
        x = self.top()
        self.hpd.add(self.ls[-1][1])
        self.ls.pop()
        return x

    def top(self):
        """
        :rtype: int
        """
        while self.ls[-1][1] in self.lsd:
            self.lsd.remove(self.ls[-1][1])
            self.ls.pop()
        return self.ls[-1][0]

    def peekMax(self):
        """
        :rtype: int
        """
        while -self.hp[0][1] in self.hpd:
            self.hpd.remove(-self.hp[0][1])
            heappop(self.hp)
        return -self.hp[0][0]

    def popMax(self):
        """
        :rtype: int
        """
        x = self.peekMax()
        _, nid = heappop(self.hp)
        self.lsd.add(-nid)
        return x
        


# Your MaxStack object will be instantiated and called as such:
# obj = MaxStack()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.top()
# param_4 = obj.peekMax()
# param_5 = obj.popMax()
```

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

```python
class Solution:

    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

> 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.
> For index i, the water volume of i: vol_i = min(left_max_i, right_max_i) - bar_i.
>
> 
>
>
> 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

The left_max array from left to right is always non-descending, the right_max is non-ascending.

Having such observation, we can say:

Given i < j, if left_max_i <= right_max_j: vol_i = left_max_i - bar_i, otherwise, vol_j = right_max_j - bar_j because, if left_max_i <= right_max_j: left_max_i <= right_max_j <= right_max_j-1 <= ... <= right_max_i, then min(left_max_i, right_max_i) is always left_max_i

Code is pasted.

```
def trap(self, bars):
    if not bars or len(bars) < 3:
        return 0
    volume = 0
    left, right = 0, len(bars) - 1
    l_max, r_max = bars[left], bars[right]
    while left < right:
        l_max, r_max = max(bars[left], l_max), max(bars[right], r_max)
        if l_max <= r_max:
            volume += l_max - bars[left]
            left += 1
        else:
            volume += r_max - bars[right]
            right -= 1
    return volume
```

69 Sqrt(x)

```
def mySqrt(self, x):
    l, r = 0, x
    while l <= r:
        mid = l + (r-l)//2
        if mid * mid <= x < (mid+1)*(mid+1):
            return mid
        elif x < mid * mid:
            r = mid
        else:
            l = mid + 1
```

44. Wildcard Matching

dp[n] means the substring s[:n] if match the pattern i
dp[0] means the empty string '' or s[:0] which only match the pattern '*'
use the reversed builtin because for every dp[n+1] we use the previous 'dp'

```
class Solution:
# @return a boolean
def isMatch(self, s, p):
    length = len(s)
    if len(p) - p.count('*') > length:
        return False
    dp = [True] + [False]*length
    for i in p:
        if i != '*':
            for n in reversed(range(length)):
                dp[n+1] = dp[n] and (i == s[n] or i == '?')
        else:
            for n in range(1, length+1):
                dp[n] = dp[n-1] or dp[n]
        dp[0] = dp[0] and i == '*'
    return dp[-1]

```

269. Alien Dictionary

There is a new alien language which uses the latin alphabet. However, the order among letters are unknown to you. You receive a list of non-empty words from the dictionary, where words are sorted lexicographically by the rules of this new language. Derive the order of letters in this language.

Example 1:

Input:
[
"wrt",
"wrf",
"er",

"ett",
"rftt"
]

Output: "wertf"
Example 2:

Input:
[
"z",
"x"
]

Output: "zx"
Example 3:

Input:
[
"z",
"x",
"z"
]

Output: ""

Explanation: The order is invalid, so return "".
Note:

You may assume all letters are in lowercase.
You may assume that if a is a prefix of b, then a must appear before b in the given dictionary.
If the order is invalid, return an empty string.
There may be multiple valid order of letters, return any one of them is fine.

```
def alienOrder(self, words):
    pre, suc = collections.defaultdict(set), collections.defaultdict(set)
    for pair in zip(words, words[1:]):
        for a, b in zip(*pair):
            if a != b:
                suc[a].add(b)
                pre[b].add(a)
                break
    chars = set(''.join(words))
    free = chars - set(pre)
    order = ''
    while free:
        a = free.pop()
        order += a
        for b in suc[a]:
            pre[b].discard(a)
```

```
            if not pre[b]:
                free.add(b)
    return order * (set(order) == chars)
```

158. Read N Characters Given Read4 II - Call multiple times

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.

Note:
The read function may be called multiple times.

Example 1:

Given buf = "abc"
read("abc", 1) // returns "a"
read("abc", 2); // returns "bc"
read("abc", 1); // returns ""
Example 2:

Given buf = "abc"
read("abc", 4) // returns "abc"
read("abc", 1); // returns ""

```
# The read4 API is already defined for you.
# @param buf, a list of characters
# @return an integer
# def read4(buf):

class Solution(object):
    def __init__(self):
        self.queue = []
    def read(self, buf, n):
        """
        :type buf: Destination buffer (List[str])
        :type n: Maximum number of characters to read (int)
        :rtype: The number of characters read (int)
        """
        idx = 0
        while True:
            buf4 = [""]*4
            read4(buf4)
            self.queue.extend(buf4)
            curr = min(len(self.queue), n - idx)
            for i in range(curr):
```

```
                buf[idx] = self.queue.pop(0)
                idx += 1
            if curr == 0:
                break
        return idx
```

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.

```
class Solution:
    _dp = [0]
    def numSquares(self, n):
        """
        :type n: int
        :rtype: int
        """

        dp = self._dp
        while len(dp) <= n:
            dp += min(dp[-i*i] for i in range(1, int(len(dp)**0.5+1))) + 1,
        return dp[n]
```

349. Intersection of Two Arrays

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.

```python
class Solution:
    def intersection(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: List[int]
        """
        set1 = set(nums1)
        set2 = set(nums2)
        return list(set1 & set2)
```

127. Word Ladder

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.

```python
class Solution(object):
    def ladderLength(self, beginWord, endWord, wordList):

        def construct_dict(word_list):
            d = {}
            for word in word_list:
                for i in range(len(word)):
                    s = word[:i] + "_" + word[i+1:]
                    d[s] = d.get(s, []) + [word]
            return d

        def bfs_words(begin, end, dict_words):
            queue, visited = [(begin, 1)], set()
            while queue:
                word, steps = queue.pop(0)
                if word not in visited:
                    visited.add(word)
                    if word == end:
                        return steps
                    for i in range(len(word)):
                        s = word[:i] + "_" + word[i+1:]
                        neigh_words = dict_words.get(s, [])
                        for neigh in neigh_words:
                            if neigh not in visited:
                                queue.append((neigh, steps + 1))
            return 0

        d = construct_dict(set(wordList))
        return bfs_words(beginWord, endWord, d)
```

251. Flatten 2D Vector

Implement an iterator to flatten a 2d vector.

Example:

Input: 2d vector =
[
[1,2],
[3],
[4,5,6]
]
Output: [1,2,3,4,5,6]
Explanation: By calling next repeatedly until hasNext returns false,
the order of elements returned by next should be: [1,2,3,4,5,6].

```python
class Vector2D(object):

    def __init__(self, vec2d):
        """
        Initialize your data structure here.
        :type vec2d: List[List[int]]
        """
        self.col = 0
        self.row = 0
        self.vec = vec2d

    def next(self):
        """
        :rtype: int
        """
        result = self.vec[self.row][self.col]
        self.col += 1
        return result

    def hasNext(self):
        """
        :rtype: bool
        """
        while self.row < len(self.vec):
            if self.col < len(self.vec[self.row]):
                return True

            self.col = 0
            self.row += 1
        return False

# Your Vector2D object will be instantiated and called as such:
# i, v = Vector2D(vec2d), []
# while i.hasNext(): v.append(i.next())
```

694. Number of Distinct Islands

Given a non-empty 2D array grid of 0's and 1's, an island is a group of 1's (representing land) connected 4-directionally (horizontal or vertical.) You may assume all four edges of the grid are surrounded by water.

Count the number of distinct islands. An island is considered to be the same as another if and only if one island can be translated (and not rotated or reflected) to equal the other.

Example 1:
11000
11000
00011
00011
Given the above grid map, return 1.

Example 2:
11011
10000
00001
11011
Given the above grid map, return 3.

Notice that:
11
1
and
1
11
are considered different island shapes, because we do not consider reflection / rotation.
Note: The length of each dimension in the given grid does not exceed 50.

```
class Solution(object):
    def numDistinctIslands(self, grid):
        """
        :type grid: List[List[int]]
        :rtype: int
        """
        island_shapes = set()
        rows, cols = len(grid), len(grid[0])
        def dfs(i, j, positions, rel_pos):
            grid[i][j] = -1
            for direction in ((0, 1), (1, 0), (-1, 0), (0, -1)):
                next_i, next_j = i + direction[0], j + direction[1]
                if (0 <= next_i < rows and 0 <= next_j < cols) and grid[next_i][next_j] ==
1:
                    new_rel_pos = (rel_pos[0] + direction[0], rel_pos[1] + direction[1])
                    positions.append(new_rel_pos)
                    dfs(next_i, next_j, positions, new_rel_pos)
        for i in range(rows):
            for j in range(cols):
                if grid[i][j] == 1:
                    positions = []
                    dfs(i, j, positions, (0, 0))
                    island_shapes.add(tuple(positions))
        return len(island_shapes)
```

68. Text Justification

Given an array of words and a width maxWidth, format the text such that each line has exactly maxWidth characters and is fully (left and right) justified.

You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces ' ' when necessary so that each line has exactly maxWidth characters.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line do not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

For the last line of text, it should be left justified and no extra space is inserted between words.

Note:

A word is defined as a character sequence consisting of non-space characters only.
Each word's length is guaranteed to be greater than 0 and not exceed maxWidth.
The input array words contains at least one word.
Example 1:

Input:
words = ["This", "is", "an", "example", "of", "text", "justification."]
maxWidth = 16
Output:
[
"This is an",
"example of text",
"justification. "
]
Example 2:

Input:
words = ["What","must","be","acknowledgment","shall","be"]
maxWidth = 16
Output:
[
"What must be",
"acknowledgment ",
"shall be "
]
Explanation: Note that the last line is "shall be " instead of "shall be",
because the last line must be left-justified instead of fully-justified.
Note that the second line is also left-justified becase it contains only one word.
Example 3:

Input:
words = ["Science","is","what","we","understand","well","enough","to","explain",
"to","a","computer.","Art","is","everything","else","we","do"]
maxWidth = 20
Output:
[
"Science is what we",
"understand well",
"enough to explain to",

"a computer. Art is",
"everything else we",
"do "
]

```
def fullJustify(self, words, maxWidth):
    res, cur, num_of_letters = [], [], 0
    for w in words:
        if num_of_letters + len(w) + len(cur) > maxWidth:
            for i in range(maxWidth - num_of_letters):
                cur[i%(len(cur)-1 or 1)] += ' '
            res.append(''.join(cur))
            cur, num_of_letters = [], 0
        cur += [w]
        num_of_letters += len(w)
    return res + [' '.join(cur).ljust(maxWidth)]
```

How does it work? Well in the question statement, the sentence "Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line do not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right" was just a really long and awkward way to say round robin. The following line implements the round robin logic:

for i in range(maxWidth - num_of_letters):
cur[i%(len(cur)-1 or 1)] += ' '
What does this line do? Once you determine that there are only k words that can fit on a given line, you know what the total length of those words is num_of_letters. Then the rest are spaces, and there are (maxWidth - num_of_letters) of spaces. The "or 1" part is for dealing with the edge case len(cur) == 1.