1. Two Sum

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].
```

Resource:

Post (https://leetcode.com/discuss/interview-experience/360829/amazon-sde1-seattle-july-2019-offer): Design question resource with experience

Leadership Principles:

Mediam Blog (https://medium.com/@scarletinked/are-you-the-leader-were-looking-for-interviewing-at-amazon-8301d787815d) Amazon LP (https://www.amazon.jobs/en/principles)

1. If your team has stopped being innovative, how would you change it?
2. How would you work with team mates of different workstyles?
3. What is important to you that the company does to have you work for them?

Other problems

Whac-A-mole (https://leetcode.com/discuss/interview-question/350139/google-phone-screen-whac-a-mole): Use Sliding Windows Algorithm to find `max_sofar` from left and right, `max([left_lst+right_lst])` would be maximum of max_sofar upto that index

Should `A` and `B` contain `max_sofar` like this?

`O=[0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0]` original

`A=[x, x, x, x, 2, 3, 3, 3, 4, 4, 4, 4, 4]` max of [i:i+5] sliding from the left

`B=[4, 4, 4, 4, 3, 3, 3, 2, x, x, x, x, x]` max of [i-5:i] sliding from the right

`S=[4, 4, 4, 4, 5, 6, 6, 5, 4, 4, 4, 4, 4]` sum

Salary Adjustment (https://leetcode.com/discuss/interview-question/351313/Google-or-Phone-Screen-or-Salary-Adjustment)

- Sort - [100, 200, 300, 400]
- Replace maximum value with k, [100, 200, 300, k]
- Calculate k , k = (800-100-200-300) = 200

- If k is less than the next max, Repeat from step 2 , else done.
    - [100, 200, k, k]
    - Calculate k , k = (800-100-200)/2 = 250
    - Done..

```
salaries=[100,200,300,400,500,400,300,200,100] # 2500
budget=2400

def salary_adjustment(salaries,budget):
    salaries.sort(reverse=True)
    for i in range(len(salaries)-1):
           avg=(budget-sum(salaries[i+1:]))/(i+1)
           if avg>=salaries[i+1]:
               break
    return avg
print(salary_adjustment(salaries,budget))
```

Interesting String (https://leetcode.com/discuss/interview-question/350312/Google-or-Onsite-or-Interesting-String)

Sum to 100 (https://leetcode.com/discuss/interview-question/357345/Uber-or-Phone-Screen-or-Sum-to-100)

Remove Extra Edge (https://leetcode.com/discuss/interview-question/358676/Google-or-Remove-Extra-Edge)

Treasure Island (https://leetcode.com/discuss/interview-question/347457/Amazon-or-OA-2019-or-Treasure-Island)

Breaking Through Walls (https://leetcode.com/discuss/interview-question/353827/Google-or-Onsite-or-Shortest-Path-Breaking-Through-Walls)

Performance issue Python Perfomance Tips (https://wiki.python.org/moin/PythonSpeed/PerformanceTips)
`s.replace('.','',1).isdigit()` is the fastest way (https://stackoverflow.com/questions/354038/how-do-i-check-if-a-string-is-a-number-float) to check if a string is number

15. 3Sum

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]
]
```

I can use 2 pointers on this

33. Search in Rotated Sorted Array

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
```

Better solution:

1. Concatnate list like this lst=lst[:]+lst[:]
2. Find start point (between min and max)
3. Search normally with offset (start point)

34. Find First and Last Position of Element in Sorted Array

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]
```

Lesson Learned:

1. Change comparison condition of binary search to find either left bound or right bound

```
class Solution:
    def searchRange(self, nums: List[int], target: int) -> List[int]:
        # This is used to find right bound
        def binary_search_right(target):
            l=0
            r=len(nums)
            while(l<r):
                mid=(l+r)//2
                if nums[mid]>target:
                    r=mid
                else:
                    l=mid+1
                print(l,r)
            return l-1
        # This is used to find left bound
        def binary_search_left(target):
            l=0
            r=len(nums)
            while(l<r):
                mid=(l+r)//2
                if nums[mid]>=target:
                    r=mid
                else:
                    l=mid+1
            return r
        left,right=binary_search_left(target),binary_search_right(target)
        print(left,right)
        return [left,right] if left<=right else [-1,-1]
```

35. Search Insert Position

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
```

Binary search

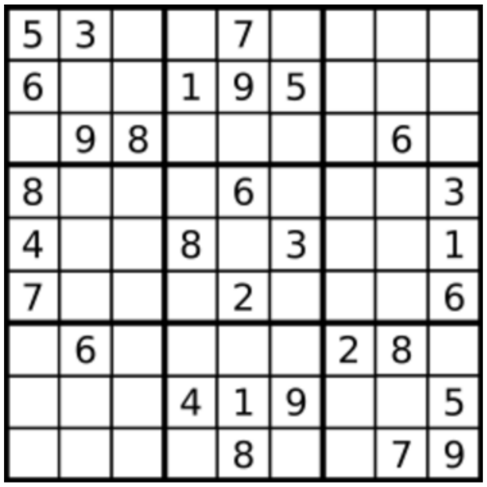
37. Sudoku Solver

Write a program to solve a Sudoku puzzle by filling the empty cells.

A sudoku solution must satisfy **all of the following rules**:

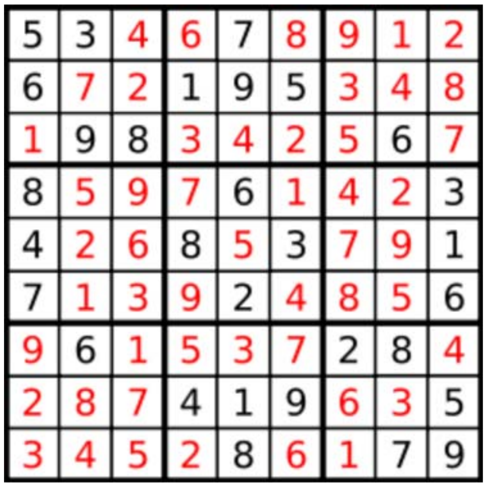
1. Each of the digits `1-9` must occur exactly once in each row.
2. Each of the digits `1-9` must occur exactly once in each column.
3. Each of the the digits `1-9` must occur exactly once in each of the 9 `3x3` sub-boxes of the grid.

Empty cells are indicated by the character `'.'` .



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 sudoku puzzle...



5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

...and its solution numbers marked in red.

Note:

- The given board contain only digits `1-9` and the character `'.'`.
- You may assume that the given Sudoku puzzle will have a single unique solution.
- The given board size is always `9x9`.

---

my post (https://leetcode.com/problems/sudoku-solver/discuss/377363/python-3-simple-solution-easy-to-understand)

note: `is_valid1()` is much faster than `is_valid2()` which means `in` is very fast.

```
def is_valid1(i, j, val):
        # check row
        if val in board1[i]: return False
        # check col
        if val in [board1[r][j] for r in range(9)]: return False
        # check block
        grp_r, grp_c = i//3, j//3
        for r in range(3):
                for c in range(3):
                        if board1[grp_r*3 + r][grp_c*3 + c] == val: return False
        return True
def is_valid2(i, j, val):
        grp_r, grp_c = i//3, j//3
        for pos in range(9):
        # check row
                if val == board1[i][pos]: return False
                # check col
                if val == board1[pos][j]: return False
                # check block
                if val == board1[grp_r*3 + pos//3][grp_c*3 + pos%3]: return False
        return True
```

Note2: i tried to keep everything in `str` and convert `num` to `str` but it is much slower in general (1000 ms using is `is_valid2()` , 880 ms using `is_valid1()`)

Final solution:

```
class Solution:
    # 384ms, 47%
    def solveSudoku(self, board: List[List[str]]) -> None:

        # convert str to int, "." to 0
        board1 = [[int(val) if val.isdigit() else 0 for val in row ] for row in board]

        def is_valid(i, j, val):
            # check row
            if val in board1[i]: return False
            # check col
            if val in [board1[r][j] for r in range(9)]: return False
            # check block
            grp_r, grp_c = i//3, j//3
            for r in range(3):
                for c in range(3):
                    if board1[grp_r*3 + r][grp_c*3 + c] == val: return False
            return True

        def backtrack(pos=0):
            if pos == len(need): return True
            i, j = need[pos]
            for num in range(1,10):
                if is_valid(i, j, num):
                    board1[i][j] = num
                    if backtrack(pos+1): return True
            # still not valid, reset val and backtrack
            board1[i][j] = 0
        # list out those cells == 0
        need = [(i, j) for i in range(9) for j in range(9) if not board1[i][j]]
        backtrack()
        # convert int to str
        board[:]=[[str(val) for val in row] for row in board1]
```

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]
]
```

First Backtracking

```
class Solution:
    # 52ms, 99.1%
    def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:
        candidates.sort()
        lst = []
        def backtracking(nums, target, index=0, sofar=0, path=[]):
            for i in range(index, len(nums)):
                num = nums[i]
                if sofar + num == target:
                    lst.append(path + [num])
                    return
                elif sofar + num < target:
                    backtracking(candidates, target, i, sofar + num, path + [num])
                else: return

        backtracking(candidates, target)
        return lst
```

Noted that the list is sorted. The purpose of passing index is to only consider larger value (to avoid the same set all over again).

There is another way to handle this without passing down current index. It is to check `if current_num >= nums[-1]`

target - curent_num sol: post (https://leetcode.com/problems/combination-sum/discuss/16510/Python-dfs-solution.)

dp sol : post (https://leetcode.com/problems/combination-sum/discuss/16506/8-line-Python-solution-dynamic-programming-beats-86.77)

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]
]
```

There are a lot to learn here:

post (https://leetcode.com/problems/combination-sum-ii/discuss/17020/Easy-to-understand-Python-solution-(backtracking).)

```
class Solution:
    #
    def combinationSum2(self, candidates: List[int], target: int) -> List[List[int]]:
        nums = sorted(candidates)
        lst =[]
        def df(target, index=0, path=[]):
            if target < 0: return
            if target == 0:  # this is slow because
                lst.append(path)
                return
            for i in range(index, len(nums)):
                # We avoid starting with these numbers because it already exist (the previou
s)
                if i > index and  nums[i] == nums[i-1]:
                    continue
                df(target - nums[i], i+1, path + [nums[i]])
        df(target)
        return lst
```

This is the optimized code

```
class Solution:
    # 48ms 94.76%
    def combinationSum2(self, candidates: List[int], target: int) -> List[List[int]]:
        nums = sorted(candidates)
        lst =[]
        def df(target, index=0, path=[]):
            for i in range(index, len(nums)):
                # We avoid starting with these numbers because it already exist (the previou
s)
                if i > index and  nums[i] == nums[i-1]:continue
                # One of the key to make it fast. We should not return in the next recursion
(it will loop thru all of them).
                if nums[i] > target: return
                if nums[i] == target:
                    path.append(nums[i])
                    lst.append(path) # this is much faster than lst.append(path + nums[i])
                    return
                df(target - nums[i], i+1, path + [nums[i]])
        df(target)
        return lst
```

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]
]
```

One liners from Stefan post (https://leetcode.com/problems/permutations/discuss/18241/One-Liners-in-Python)

```
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        ans = []
        def re(nums, pick = []):
            if not nums:
                ans.append(pick)
                return
            for i in range(len(nums)):
                rest = nums[:i] + nums[i+1:]
                re(rest, pick + [nums[i]])
        re(nums)
        return ans
```

53. Maximum Subarray

Given an integer array `nums`, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

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 divide and conquer approach, which is more subtle.

First DP problem

This is Kadane's Algorithm. Similar to `stock purchse` problem

Still confusing: post (https://leetcode.com/problems/maximum-subarray/discuss/20194/A-Python-solution) other post (https://leetcode.com/problems/maximum-subarray/discuss/20487/Python-solutions-(DP-O(n)-space-O(1)-space).)

```
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        max_sofar = max_cur =  float('-inf')
        for num in nums:
            max_cur = max(max_cur + num, num)
            max_sofar = max(max_sofar, max_cur)
        return max_sofar
```

using accumulate 1liner. accumulate's 2nd arg is `operator.add` by default

```
from itertools import accumulate
def maxSubArray(self, nums: List[int]) -> int:
    return max(accumulate(nums, lambda x, y: max(y, x+y) ))
```

54. Spiral Matrix

Given a matrix of *m* x *n* elements (*m* rows, *n* columns), return all elements of the matrix in spiral order.

Example 1:

```
Input:
[
 [ 1, 2, 3 ],
 [ 4, 5, 6 ],
 [ 7, 8, 9 ]
]
Output: [1,2,3,6,9,8,7,4,5]
```

Example 2:

```
Input:
[
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9,10,11,12]
]
Output: [1,2,3,4,8,12,11,10,9,5,6,7]
```

Very good approach from StefanPochmann post (https://leetcode.com/problems/spiral-matrix/discuss/20571/1-liner-in-Python-%2B-Ruby)

Learn:

- `zip(*lst)` can be very useful to create a new matrix
- Again, `lst.append()` significantly decrease processed time

```
class Solution:
    # Revised, use the same approach as Stefan
    # 32 ms, 91.77%
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
        ans = []
        def spiral(matrix):
            if matrix and matrix[0]:
                ans.extend(matrix[0])
                new_matrix = [row[::-1] for row in matrix[1:]]
                spiral(list(zip(*new_matrix)))
        spiral(matrix)
        return ans
    # Initial solution
        # 40 ms 37.48%
    def spiralOrder1(self, matrix: List[List[int]]) -> List[int]:
        ans = []
        def spiral(matrix):
            if not matrix: return
            R, C = len(matrix), len(matrix[0])
            if 0 in (R,C): return
            ans.extend(matrix[0][:-1])
            ans.extend([row[-1] for row in matrix])
            if C > 1 and R > 1:
                ans.extend(reversed(matrix[-1][:-1]))
            if R > 1 and C > 1:
                ans.extend([row[0] for row in reversed(matrix[1:-1])])
            new_matrix = [[i for i in row[1:-1]]
                          for row in matrix[1:-1]]
            spiral(new_matrix)

        spiral(matrix)
        return ans
```

55. Jump Game

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.
```

```
class Solution:
    # dp, 100 ms 84%, backward
    def canJump(self, nums: List[int]) -> bool:
        goal = len(nums)-1
        for i in reversed(range(len(nums))):
            if i + nums[i] >= goal:
                goal = i
        return not goal

    # dp, 100 ms 84%, forward
    def canJump(self, nums: List[int]) -> bool:
        m = 0
        for i, num in enumerate(nums):
            if i > m: return False
            m = max(m, i+num)
        return m >= len(nums)-1

    # original, 100ms, 84%
    def canJump1(self, nums: List[int]) -> bool:
        if set(nums)== {1} : return True
        nums = nums[::-1]
        self.possible = False

        def f(nums, target):
            # print(target, nums)
            if not target:
                self.possible =True
                return True
            if not nums:
                return False
            for i in range(len(nums)):
                val = nums[i]
                if val > i:
                    #good
                    return f(nums[i+1:], target - (i+1))
            return False
        f(nums[1:], len(nums)-1)
        return self.possible
```

58. Length of Last Word

Given a string *s* consists of upper/lower-case alphabets and empty space characters `' '`, return the length of last word in the string.

If the last word does not exist, return 0.

Note: A word is defined as a character sequence consists of non-space characters only.

Example:

```
Input: "Hello World"
Output: 5
```

Lesson learned:

1. `str.split()` default is `space`
2. `str.split()` splits multiple spaces like this `"  "`

59. Spiral Matrix II

Given a positive integer *n*, generate a square matrix filled with elements from 1 to n^2 in spiral order.

Example:

```
Input: 3
Output:
[
 [ 1, 2, 3 ],
 [ 8, 9, 4 ],
 [ 7, 6, 5 ]
]
```

Stefan 3 sols: post (https://leetcode.com/problems/spiral-matrix-ii/discuss/22282/4-9-lines-Python-solutions)

Note: `[[0]]*10` will give the same result but the list are not distinct instances,they are just n references to the same instance, more here (https://stackoverflow.com/questions/17702937/generating-sublists-using-multiplication-unexpected-behavior)

To try: 9,8,...,2,1 using `map, zip, ...`

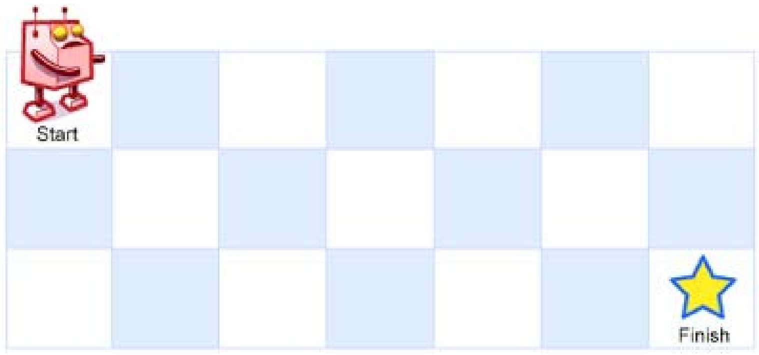
```
class Solution:
    # Better solution using di, dj 36ms 85%
    def generateMatrix(self, n: int) -> List[List[int]]:
        matrix = [[0 for _ in range(n)] for _ in range(n)]
        i, j, di, dj = 0, 0, 0, 1
        for val in range(1,n**2+1):
            matrix[i][j] = val
            if matrix[(i+di) % n][(j+dj) % n]:
                dj, di = -di, dj
            i += di
            j += dj
        return matrix

    # ugly sol 80 ms
    def generateMatrix1(self, n: int) -> List[List[int]]:
        matrix = [[0 for _ in range(n)] for _ in range(n)]
        def fill(n, val=1, layer=0):
            if n<=0: return
            if not layer:
                matrix[0]= list(range(val,val + n))
                matrix[-1]= list(range((n-1)*3 + 1, (n-1)*2, -1))
            else:
                matrix[layer][layer:-layer]= list(range(val,val + n))
                matrix[-layer-1][layer:-layer]= list(range((n-1)*3 + val, (n-1)*2 - 1 + val,
-1))
            tmp = n + val
            for row in matrix[1+layer:-1-layer]:
                row[-1-layer] += tmp
                tmp +=1
            tmp = (n-1)*4 + val -1
            for row in matrix[1+layer:-1-layer]:
                row[layer] += tmp
                tmp -=1
            fill(n-2, (n-1)*4 + val, layer+1)
        fill(n)
        return matrix
```

62. Unique Paths

A robot is located at the top-left corner of a *m* x *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 x 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 -> Right -> Down
2. Right -> Down -> Right
3. Down -> Right -> Right
```

Example 2:

```
Input: m = 7, n = 3
Output: 28
```

A bit of statistic:

```
    def uniquePaths(self, m: int, n: int) -> int:
            if not m or not n:
                    return 0
            return int(math.factorial(m+n-2)/(math.factorial(n-1) * math.factorial(m-1)))

    def uniquePaths1(self, m: int, n: int) -> int:
            import numpy as np
            A = np.ones((m,n))
            for r in range(1,m):
                    for c in range(1,n):
                            A[r][c] = A[r-1][c] + A[r][c-1]
            return int(A[-1][-1])
```

65. Valid Number

Validate if a given string can be interpreted as a decimal number.

Some examples:

```
"0" => true
" 0.1 " => true
"abc" => false
"1 a" => false
"2e10" => true
" -90e3   " => true
" 1e" => false
"e3" => false
" 6e-1" => true
" 99e2.5 " => false
"53.5e93" => true
" --6 " => false
"-+3" => false
"95a54e53" => false
```

Note: It is intended for the problem statement to be ambiguous. You should gather all requirements up front before implementing one. However, here is a list of characters that can be in a valid decimal number:

- Numbers 0-9
- Exponent - "e"
- Positive/negative sign - "+"/"-"
- Decimal point - "."

Of course, the context of these characters also matters in the input.

Update (2015-02-10):
The signature of the `C++` function had been updated. If you still see your function signature accepts a `const char *` argument, please click the reload button to reset your code definition.

Check valid number using regex

66. Plus One

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.
```

Good way of using map, int, str

```
class Solution:
    def plusOne(self, digits: List[int]) -> List[int]: #40ms, 60%
        return map(int,str(int("".join(map(str, digits))) + 1)) if digits else [1]

    def plusOne2(self, digits: List[int]) -> List[int]:#40ms, 60%
        num = sum([10**i * num for i, num in enumerate(reversed(digits))]) + 1
        return list(str(num))

    def plusOne1(self, digits: List[int]) -> List[int]: #36ms, 86%
        def sm(arr, c=1):
            if not arr:
                return [c] if c else []
            total = arr[-1] + c
            if total < 10:
                return sm(arr[:-1], 0) + [total % 10]
            return sm(arr[:-1], 1) + [total % 10]
        return sm(digits)
```

67. Add Binary

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"
```

An example (https://leetcode.com/problems/add-binary/discuss/24500/An-accepted-concise-Python-recursive-solution-10-lines) of recursive call inside recursive call (nested recursive)

```
class Solution:
    def addBinary(self, a, b):
        if len(a)==0: return b
        if len(b)==0: return a
        if a[-1] == '1' and b[-1] == '1':
            return self.addBinary(self.addBinary(a[0:-1],b[0:-1]),'1')+'0'
        if a[-1] == '0' and b[-1] == '0':
            return self.addBinary(a[0:-1],b[0:-1])+'0'
        else:
            return self.addBinary(a[0:-1],b[0:-1])+'1'
```

70. Climbing Stairs

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 climb to the top?

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
```

This is fibonacci sequence

```
class Solution:
    # second realization 36 ms
    def climbStairs(self, n: int) -> int:
        a = 1
        b = 1
        for _ in range(n-1):
            a, b = b, a + b
        return b

    # shortened version of initial sol 32 ms
    def climbStairs2(self, n: int) -> int:
        lst = [1]
        for _ in range(n-1):
            lst.append(sum(lst[: -1]) + 2)
        return lst[-1]

    # based on initial realization of the connections of the numbers, 40 ms
    def climbStairs1(self, n: int) -> int:
        d = {}
        def recursive(n):
            if n == 1: return 1
            total = 0
            for i in range(1, n-1):
                if i in d:
                    val = d[i]
                else:
                    val = recursive(i)
                    d[i] = val
                total += val
            return total + 2
        return recursive(n)
```

73. Set Matrix Zeroes

Given a *m* x *n* matrix, if an element is 0, set its entire row and column to 0. Do it **in-place** (https://en.wikipedia.org/wiki/In-place_algorithm).

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]
]
```

Follow up:

- A straight forward solution using O(*mn*) space is probably a bad idea.
- A simple improvement uses O(*m* + *n*) space, but still not the best solution.
- Could you devise a constant space solution?

There is another that has space O(1). That is to store row and col index inplace at the first row/col. Noted that it's necessery to have variable for `matrix[0][0]`

```
class Solution:

    # 148 ms 92%.
    # Use list to store col and row
    def setZeroes(self, matrix: List[List[int]]) -> None:
        if not matrix: return []
        R, C = len(matrix), len(matrix[0])
        r_lst, c_lst = [], []
        # Store col, row index
        for r in range(R):
            for c in range(C):
                if not matrix[r][c]:
                    c_lst.append(c)
                    r_lst.append(r)
        # Change val
        for r in r_lst:
            matrix[r][:] = [0]*C
        for c in c_lst:
            for r in range(R):
                matrix[r][c] = 0

    # Init sol
    def setZeroes1(self, matrix: List[List[int]]) -> None:
        """
        Do not return anything, modify matrix in-place instead.
        """
        if not matrix: return []

        def change_to_NAN(i, j):
            for r in range(R):
                if not matrix[r][j]: break
                matrix[r][j] = "0"
            for c in range(C):
                if not matrix[i][c]: break
                matrix[i][c] = "0"

        R, C = len(matrix), len(matrix[0])
        for i in range(R):
            for j in range(C):
                if not matrix[i][j]:
                    matrix[i][j] = "0"
                    change_to_NAN(i, j)
        for i in range(R):
            for j in range(C):
                if matrix[i][j] == "0":
                    matrix[i][j] = 0
```

94. Binary Tree Inorder Traversal

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

Example:

```
Input: [1,null,2,3]
   1
    \
     2
    /
   3

Output: [1,3,2]
```

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

---

2 ways to do inorderTraversal:

- dsf(left) then add cur.val then dsf(right)
- use stack

```
class Solution:
    # DSF
    def inorderTraversal(self, node: TreeNode) -> List[int]:
        lst=[]
        def dsf(node):
            if node:
                if node.left:
                    dsf(node.left)
                lst.append(node.val)
                if node.right:
                    dsf(node.right)

        dsf(node)
        return lst

    # inorderTraversal
    def inorderTraversal1(self, node: TreeNode) -> List[int]:
        stack=[]
        ans=[]
        while(stack or node):
            while(node):
                stack.append(node)
                node=node.left
            node=stack.pop()
            ans.append(node.val)
            node=node.right
        return ans
```

98. Validate Binary Search Tree

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.
```

Lesson Learned:

Inorder Traversal can be used to check validBST In this case, DSF is faster than BSF

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def isValidBST(self, root: TreeNode) -> bool:
        # inorder_check()
        stack=[]
        inorder=float('-inf')
        while(stack or root):
            while(root):
                stack.append(root)
                root=root.left
            root=stack.pop()
            # inorder should always be greater than current val
            if root.val <= inorder:
                return False
            inorder=root.val
            root=root.right
        return True

    def isValidBST1(self, root: TreeNode) -> bool:
        def check_tree(node,low,high):
            if node==None: return True
            if node.left:
                if node.left.val <= low or node.left.val >= node.val:
                    return False
            if node.right:
                if node.right.val <= node.val or node.right.val >= high:
                    return False
            # Change low and high for the next node
            return check_tree(node.left,low,node.val)\
                and check_tree(node.right,node.val,high)
        from math import inf
        return check_tree(root,float(-inf),float(inf))
```

100. Same Tree

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
```

Lesson Learned:

Check if 2 Binary trees are the same:

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def isSameTree(self, p: TreeNode, q: TreeNode) -> bool:
        if p is None or q is None:
            return p is q
            # same as:
            # if p==q: return True
            # else: return False
        return p.val==q.val and self.isSameTree(p.left,q.left) \
                            and self.isSameTree(p.right,q.right)
```

More ways (stack, queue): post (https://leetcode.com/problems/same-tree/discuss/32894/Python-Recursive-solution-and-DFS-Iterative-solution-with-stack-and-BFS-Iterative-solution-with-queue)

104. Maximum Depth of Binary Tree

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.

Lesson Learned:

Max depth in binary Tree:

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def maxDepth(self, root: TreeNode) -> int:
        def count(node):
            if node==None: return 0
            return 1+ max([count(node.left),count(node.right)])
        return count(root)
```

111. Minimum Depth of Binary Tree

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]`,

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

return its minimum depth = 2.

Lesson Learned:

Binary Tree Min Depth:

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def minDepth(self, node: TreeNode) -> int:
        if node is None: return 0
        if node.left is None or node.right is None:
            return  1 + self.minDepth(node.left) + self.minDepth(node.right)
        return 1 + min(self.minDepth(node.left),self.minDepth(node.right))
```

112. Path Sum

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`,

```
      5
     / \
    4   8
   /   / \
  11  13  4
 /  \      \
7    2      1
```

return true, as there exist a root-to-leaf path `5->4->11->2` which sum is 22.

Lesson Learned:

2 methods:

```
class Solution:
    # sum -= root.val until sum = root.val of leaf node.
        def hasPathSum(self, root: TreeNode, sum: int) -> bool:
        if root is None: return None
        if root.left==root.right==None and root.val==sum:
            return True
        sum-=root.val
        return self.hasPathSum(root.left, sum) or self.hasPathSum(root.right, sum)

        # make a list and append value to it
    def hasPathSum2(self, root: TreeNode, sum: int) -> bool:
        def recursive_path(root):
            if root is None: return []
            if root.left==root.right==None: return [root.val]
            return [root.val + i for i  in recursive_path(root.left) ]\
                    +[root.val + i for i  in recursive_path(root.right) ]
        lst=recursive_path(root)
        return sum in lst
```

Take a look at `return root`. This will exit recursion if root is None, else continue.

```
class Solution:
    def rangeSumBST(self, root: TreeNode, L: int, R: int) -> int:
        return root and root.val * (L <= root.val <= R) + self.rangeSumBST(root.left, L, R) +
self.rangeSumBST(root.right, L, R) or 0
```

113. Path Sum II

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`,

```
      5
     / \
    4   8
   /   / \
  11  13  4
 /  \    / \
7    2  5   1
```

Return:

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

many more ways (https://leetcode.com/problems/path-sum-ii/discuss/36829/Python-solutions-(Recursively-BFS%2Bqueue-DFS%2Bstack)) by caikehe

my sol:

```
class Solution:
    ## 44ms, 99%
    def pathSum(self, root: TreeNode, sum: int) -> List[List[int]]:
        if not root: return []
        ans = []
        def recurse(node, target, sofar):
            if node:
                if node.left is node.right is None and target == node.val:
                    ans.append(sofar + [node.val])
                    return
                recurse(node.left, target - node.val, sofar + [node.val])
                recurse(node.right, target - node.val, sofar + [node.val])

        recurse(root, sum, [])
        return ans
```

120. Triangle

Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below.

For example, given the following triangle

```
[
     [2],
    [3,4],
   [6,5,7],
  [4,1,8,3]
]
```

The minimum path sum from top to bottom is `11` (i.e., **2** + **3** + **5** + **1** = 11).

Note:

Bonus point if you are able to do this using only *O*(*n*) extra space, where *n* is the total number of rows in the triangle.

Notice the use of `reduce`

```
class Solution:
    # bottom up, 1 liner, 64ms - 72 ms, 94% - 56%
    def minimumTotal(self, triangle: List[List[int]]) -> int:
        from functools import reduce
        def combines_rows(lower_row, upper_row):
            return [upper_row[i] + min(lower_row[i], lower_row[i+1])
                   for i in range(len(upper_row))]
        return reduce(combines_rows, triangle[::-1])[0]



    # bottom up, 64ms - 72 ms, 94% - 56%
    def minimumTotal2(self, triangle: List[List[int]]) -> int:
        for r in reversed(range(len(triangle)-1)):
            prev = triangle[r + 1]
            curr = triangle[r]
            for i in range(len(curr)):
                curr[i] += min(prev[i], prev[i+1])
        return triangle[0][0]

    # top down, 68ms 80%
    def minimumTotal1(self, triangle: List[List[int]]) -> int:
        for r in range(1, len(triangle)):
            prev = triangle[r - 1]
            row = triangle[r]
            for i in range(len(row)):
                if i == 0: row[i] += prev[i]
                elif i == len(row)-1: row[i] += prev[i-1]
                else: row[i] += min(prev[i-1], prev[i])
        return min(triangle[-1])
```

123. Best Time to Buy and Sell Stock III

Say you have an array for which the i^{th} element is the price of a given stock on day *i*.

Design an algorithm to find the maximum profit. You may complete at most *two* transactions.

Note: You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again).

Example 1:

```
Input: [3,3,5,0,0,3,1,4]
Output: 6
Explanation: Buy on day 4 (price = 0) and sell on day 6 (price = 3), profit = 3-0 = 3.
             Then buy on day 7 (price = 1) and sell on day 8 (price = 4), profit = 4-1 = 3.
```

Example 2:

```
Input: [1,2,3,4,5]
Output: 4
Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4.
             Note that you cannot buy on day 1, buy on day 2 and sell them later, as you are
             engaging multiple transactions at the same time. You must sell before buying again
```

Example 3:

```
Input: [7,6,4,3,1]
Output: 0
Explanation: In this case, no transaction is done, i.e. max profit = 0.
```

Lesson Learned: Kadane's Algorithm (https://leetcode.com/problems/best-time-to-buy-and-sell-stock/discuss/39038/Kadane's-Algorithm-Since-no-one-has-mentioned-about-this-so-far-%3A) or `maximum subarray problem`

```
class Solution:
    def maxProfit(self, prices: List[int]) -> int:

        import math
        buy1=buy2=math.inf
        buy1sell1=buy2sell2=0
        for p in prices:
            # similar to 121. Switch min if found smaller
            # else we keep cummulate to find max_profit
            buy1=min(buy1,p)
            buy1sell1=max(buy1sell1,p-buy1)

            # after the first one, we have a profit of buy1sell1.
            # This is simply a shift in value, then we repeat the same thing.
            buy2=min(buy2,p-buy1sell1)
            buy2sell2=max(buy2sell2,p-buy2)

        return buy2sell2
```

130. Surrounded Regions

Given a 2D board containing `'X'` and `'O'` (**the letter O**), capture all regions surrounded by `'X'`.

A region is captured by flipping all `'O'` s into `'X'` s in that surrounded region.

Example:

```
X X X X
X O O X
X X O X
X O X X
```

After running your function, the board should be:

```
X X X X
X X X X
X X X X
X O X X
```

Explanation:

Surrounded regions shouldn't be on the border, which means that any `'O'` on the border of the board are not flipped to `'X'`. Any `'O'` that is not on the border and it is not connected to an `'O'` on the border will be flipped to `'X'`. Two cells are connected if they are adjacent cells connected horizontally or vertically.

Stefan (https://leetcode.com/problems/surrounded-regions/discuss/41630/9-lines-Python-148-ms)

`'XO'[c == 'N']` = O if c=N else X

```
class Solution:
    def solve(self, board: List[List[str]]) -> None:
        """
        Do not return anything, modify board in-place instead.
        """
        if not board: return
        R,C=len(board),len(board[0])

        def dfs(r,c):
            if 0<=r<R and 0<=c<C and board[r][c]=="O":
                board[r][c]="N"
                dfs(r+1,c)
                dfs(r-1,c)
                dfs(r,c+1)
                dfs(r,c-1)
        # first and last row
        for r in range(R):
            for c in [0,C-1]:
                if board[r][c] == "O": dfs(r,c)

                # first and last column
        for c in range(1,C-1):
            for r in [0,R-1]:
                if board[r][c] == "O": dfs(r,c)

        board[:] = [['XO'[c == 'N'] for c in row] for row in board]
                # is faster than the following:
        # for r in range(R):
        #     for c in range(C):
        #         if board[r][c] == "O": board[r][c]="X"
        #         if board[r][c] == "N": board[r][c]="O"
```

145. Binary Tree Postorder Traversal

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?

Good Article in #230 (https://leetcode.com/articles/kth-smallest-element-in-a-bst/)

1. preorder DFS
2. inorder DFS
3. postorder DFS
4. BFS has to be done interation with queue

My sols:

```
    # iteration
    def postorderTraversal(self, node: TreeNode) -> List[int]:
        stack, lst=[node], []
        while (stack):
            node = stack.pop()
            if node:
                lst.append(node.val)
                stack.extend([node.left,node.right])
        return reversed(lst)

    # recursion
    def postorderTraversal1(self, node: TreeNode) -> List[int]:
        return self.postorderTraversal(node.left)\
            + self.postorderTraversal(node.right)\
            + [node.val] if node else []
```

3 ways to do recursive traversal

For post-order traversal:

```
def postorder(r):
            return postorder(r.left)  + postorder(r.right) + [r.val] if r else []
```

For in-order traversal:

```
def inorder(r):
            return inorder(r.left) + [r.val] + inorder(r.right) if r else []
```

For preorder traversal:

```
def inorder(r):
            return [r.val] + preorder(r.left) + preorder(r.right) if r else []
```

3 ways to do iteration traversal

For post-order traversal:

```
def postorderTraversal(self, root):
    res, stack = [], [(1, root)]
    while stack:
        p = stack.pop()
        if not p[1]: continue
        stack.extend([(0, p[1]), (1, p[1].right), (1, p[1].left)]) if p[0] != 0 else res.appe
nd(p[1].val)
    return res
```

For in-order traversal:

```
def inorderTraversal(self, root):
    res, stack = [], [(1, root)]
    while stack:
        p = stack.pop()
        if not p[1]: continue
        stack.extend([(1, p[1].right), (0, p[1]), (1, p[1].left)]) if p[0] != 0 else res.appe
nd(p[1].val)
    return res
```

For pre-order traversal:

```
def preorderTraversal(self, root):
    res, stack = [], [(1, root)]
    while stack:
        p = stack.pop()
        if not p[1]: continue
        stack.extend([(1, p[1].right), (1, p[1].left), (0, p[1])]) if p[0] != 0 else res.appe
nd(p[1].val)
    return res
```

151. Reverse Words in a String

Given an input string, reverse the string word by word.

Example 1:

```
Input: "the sky is blue"
Output: "blue is sky the"
```

Example 2:

```
Input: "  hello world!  "
Output: "world! hello"
Explanation: Your reversed string should not contain leading or trailing spaces.
```

Example 3:

```
Input: "a good   example"
Output: "example good a"
Explanation: You need to reduce multiple spaces between two words to a single space in the reve
```

Note:

- A word is defined as a sequence of non-space characters.
- Input string may contain leading or trailing spaces. However, your reversed string should not contain leading or trailing spaces.
- You need to reduce multiple spaces between two words to a single space in the reversed string.

Follow up:

For C programmers, try to solve it *in-place* in *O*(1) extra space.

Lesson Learned:

1. Use trigger signals __`space`__ to identify a word in a sentence

```
class Solution:
    def reverseWords(self, s: str) -> str:
        start_i=0
        s=" "+s+" "
        lst=[]
        for i,c in enumerate(s):
            # if prev=" " and current!=" " then it is the start of a word
            if s[i-1]==" " and c!=" ":
                start_i=i
            # if current=" " and prev!=" " then it is the end of a word
            elif c==" " and s[i-1]!=" ":
                lst.append( s[ start_i : i+1 ].strip())
        return " ".join( reversed ( lst ) )


    def reverseWords2(self, s: str) -> str:
        words=s.split()
        return " ".join(reversed(words))
```

155. Min Stack

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.

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.
```

Lesson Learned:

How to implement a stack

162. Find Peak Element

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.

Simple problem that can be solved using binary search.

Important: Pay attention to how left = mid +1

```
class Solution:
    # binary search 48 ms, 97%
    def findPeakElement(self, nums: List[int]) -> int:
        if len(nums)==1: return 0
        left, right = 0, len(nums)-1
        while left < right:
            mid = (left + right)//2
            if nums[mid + 1] > nums[mid]:
                left = mid + 1
            else:
                right = mid
        return left

    # iterative, 60 ms, 16%
    def findPeakElement1(self, nums: List[int]) -> int:
        for i in range(len(nums)):
            if i == len(nums)-1 or nums[i] > nums[i+1]:
                return i
```

167. Two Sum II - Input array is sorted

Given an array of integers that is already ***sorted in ascending order***, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2.

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 the *same* element twice.

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.
```

Lesson Learned:

1. If we have a sorted list of numbers and want to find a combination that add up to a target. We can start from the 2 ends and if sum>target: right-=1 else left+=1

```
class Solution:
 def twoSum(self, numbers: List[int], target: int) -> List[int]:
     left=0
     right=len(numbers)-1
     while (numbers[left]+numbers[right] != target):
         if numbers[left]+numbers[right] > target:
             right-=1
         else:
             left+=1
     return [left+1,right+1]
```

2. Post (https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/discuss/261140/C%2B%2B-with-explanation-Clear-and-simple)
3. We can use dictionary (hash table) like problem #1 ``` def twoSum2(self, numbers: List[int], target: int) -> List[int]:

```
 d={}
 for i,number in enumerate(numbers):
     if number not in d:
         d[target-number]=i
     else:
         return [d[number]+1,i+1]
```

```
```

189. Rotate Array

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?

There are many ways to rotate array (read sol) My way:

```
class Solution:
    """
    Do not return anything, modify nums in-place instead.
    """
    def rotate(self, nums: List[int], k: int) -> None:
        temp=[]
        k=k%len(nums)
        for num in nums[-k:]+nums[:-k]:
            temp.append(num)
        nums[:]=temp


    def rotate1(self, nums: List[int], k: int) -> None:
        from collections import deque
        d=deque(nums)
        d.rotate(k)
        nums[:]=list(d)
```

198. House Robber

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight **without alerting the police**.

Example 1:

```
Input: [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
             Total amount you can rob = 1 + 3 = 4.
```

Example 2:

```
Input: [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).
             Total amount you can rob = 2 + 9 + 1 = 12.
```

Hard dp

Self Explanation: Imgur (https://i.imgur.com/X0EHENB.jpg) Better sol: post (https://leetcode.com/problems/house-robber/discuss/55938/DP-with-two-variables-Easiest-Solution-(I-think)-O(n))

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
```

Lesson Learned:

For BFS, when we deal with deque we can do in 2 ways:

```
queue=[(r,c)]
while(queue):
    r,c=queue.pop(0)
    # do work
    queue.append(...)
```

or

```
stack=[(r,c)]
for r,c in stack:
    # do work
    stack.append(...)
```

see more in post (https://leetcode.com/problems/number-of-islands/discuss/56575/Python-iterativerecursive-BFSDFS) or really short python in post (https://leetcode.com/problems/number-of-islands/discuss/56349/7-lines-Python-~14-lines-Java) from Stefan

204. Count Primes

Count the number of prime numbers less than a non-negative number, ***n***.

Example:

```
Input: 10
Output: 4
Explanation: There are 4 prime numbers less than 10, they are 2, 3, 5, 7.
```

Fastest way to find prime numbers

1. while(1) is faster than while(True), True,False takes more memory than 0,1
2. lst[index] = O(1). Very fast
3. in dict/set =O(1). Very fast
4. `primes=[0,1] * (n//2) + [0] * (n%2!=0)` takes more memory than seperate if
5. Only need to check upto sqrt(n) because larger number already got checked off

6. `for j in range(i*i,n,i): primes[j]=0` is slower than `primes[i*i:n:i]=[0]*int((n-i*i-1)/i + 1)` but use less memory

```
class Solution:
    def countPrimes(self, n: int) -> int:
        if n<3: return 0
        def count_primes(n):

            # Generate [0,1,0,1,0,1,...].
            # Note: with this generation, 1 is prime and 2 is not prime. But I return sum of
1s, so it does not matter much
            primes=[0,1]*(n//2)
            if n%2!=0: primes+=[0] # add another one if n is odd

            for i in range(3,int(n**0.5)+1,2): # I don't need to check for even numbers
                if primes[i]:
                    primes[i*i:n:i]=[0]*int((n-i*i-1)/i + 1)
            return sum(primes)
        s=count_primes(n)
        return s
```

226. Invert Binary Tree

Invert a binary tree.

Example:

Input:

```
     4
   /   \
  2     7
 / \   / \
1   3 6   9
```

Output:

```
     4
   /   \
  7     2
 / \   / \
9   6 3   1
```

Trivia:
This problem was inspired by this original tweet (https://twitter.com/mxcl/status/608682016205344768) by Max Howell (https://twitter.com/mxcl):

> Google: 90% of our engineers use the software you wrote (Homebrew), but you can’t invert a binary tree on a whiteboard so f*** off.

Lesson Learned:

Flip Binary Tree:

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    # First method: return new root/node
        def invertTree(self, node: TreeNode) -> TreeNode:
        if node:
            node.left,node.right=self.invertTree(node.right),self.invertTree(node.left)
            return node
    # Second method: Switch node inplace
    def invertTree_inplace(self, root: TreeNode) -> TreeNode:
        def inverting(node):
            if node:
                node.left,node.right=node.right,node.left
                inverting(node.left)
                inverting(node.right)
        inverting(root)
        return root
```

Other ways (queue,deque): post (https://leetcode.com/problems/invert-binary-tree/discuss/62705/Python-solutions-(recursively-dfs-bfs).)

228. Summary Ranges

Given a sorted integer array without duplicates, return the summary of its ranges.

Example 1:

```
Input:  [0,1,2,4,5,7]
Output: ["0->2","4->5","7"]
Explanation: 0,1,2 form a continuous range; 4,5 form a continuous range.
```

Example 2:

```
Input:  [0,2,3,4,6,8,9]
Output: ["0","2->4","6","8->9"]
Explanation: 2,3,4 form a continuous range; 8,9 form a continuous range.
```

Other ways by Stefan post (https://leetcode.com/problems/summary-ranges/discuss/63193/6-lines-in-Python), does not seem to be very optimized, though

230. Kth Smallest Element in a BST

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 ≤ k ≤ BST's total elements.

Example 1:

```
Input: root = [3,1,4,null,2], k = 1
   3
  / \
 1   4
  \
   2
Output: 1
```

Example 2:

```
Input: root = [5,3,6,2,4,null,null,1], k = 3
       5
      / \
     3   6
    / \
   2   4
  /
 1
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?

257. Binary Tree Paths

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

Note: A leaf is a node with no children.

Example:

```
Input:

   1
 /   \
2     3
 \
  5

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

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

Code:

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def binaryTreePaths(self, root: TreeNode) -> List[str]:
        lst=[]
        def walk(node):
            if node is None: return []
            if node.left==node.right==None:
                return [str(node.val)]
            return [str(node.val) + "->"+ i for i in walk(node.left)] +\
                    [str(node.val) +"->"+ i for i in walk(node.right)]
        return walk(root)
```

Other ways: post (https://leetcode.com/problems/binary-tree-paths/discuss/68287/5-lines-recursive-Python)

283. Move Zeroes

Given an array `nums`, write a function to move all `0`'s to the end of it while maintaining the relative order of the non-zero elements.

Example:

```
Input: [0,1,0,3,12]
Output: [1,3,12,0,0]
```

Note:

1. You must do this **in-place** without making a copy of the array.
2. Minimize the total number of operations.

Easy problem but require deeper thinking and analysis

```
class Solution:
    # in-place
    def moveZeroes(self, nums):
        zero = 0  # records the position of "0"
        for i in range(len(nums)):
            if nums[i] != 0:
                nums[i], nums[zero] = nums[zero], nums[i]
                zero += 1

    def moveZeroes1(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        try: zero=nums.index(0)
        except: return
        for i in range(zero+1,len(nums)):
            if nums[i]:
                nums[zero], nums[i] = nums[i], nums[zero]
                zero+=1
```

287. Find the Duplicate Number

Given an array *nums* containing *n* + 1 integers where each integer is between 1 and *n* (inclusive), prove that at least one duplicate number must exist. Assume that there is only one duplicate number, find the duplicate one.

Example 1:

```
Input: [1,3,4,2,2]
Output: 2
```

Example 2:

```
Input: [3,1,3,4,2]
Output: 3
```

Note:

1. You **must not** modify the array (assume the array is read only).
2. You must use only constant, *O*(1) extra space.
3. Your runtime complexity should be less than $O(n^2)$.
4. There is only one duplicate number in the array, but it could be repeated more than once.

One of the fast ways to detect repeat in an array `in` of set is O(1) `set.add()` is O(1)

```
class Solution:
    def findDuplicate(self, nums: List[int]) -> int:
        seen=set()
        for num in nums:
            if num in seen: return num
            seen.add(num)
        return -1
```

345. Reverse Vowels of a String

Write a function that takes a string as input and reverse only the vowels of a string.

Example 1:

```
Input: "hello"
Output: "holle"
```

Example 2:

```
Input: "leetcode"
Output: "leotcede"
```

Note:
The vowels does not include the letter "y".

Lesson Learned:

1. Can use regex (https://leetcode.com/problems/reverse-vowels-of-a-string/discuss/81262/1-2-lines-PythonRuby) aswell
2. `"".join(lst)` combine with `generator` is ***faster*** than `string concatnation` combined with `loop`

383. Ransom Note

Given an arbitrary ransom note string and another string containing letters from all the magazines, write a function that will return true if the ransom note can be constructed from the magazines ; otherwise, it will return false.

Each letter in the magazine string can only be used once in your ransom note.

Note:
You may assume that both strings contain only lowercase letters.

```
canConstruct("a", "b") -> false
canConstruct("aa", "ab") -> false
canConstruct("aa", "aab") -> true
```

Lesson Learned:

1. We can do summation and substraction on `Counter` post (https://leetcode.com/problems/ransom-note/discuss/85837/O(m%2Bn)-one-liner-Python)

   ```
   def canConstruct(self, ransomNote, magazine):
    return not collections.Counter(ransomNote) - collections.Counter(magazine)
   ```

387. First Unique Character in a String

Given a string, find the first non-repeating character in it and return it's index. If it doesn't exist, return -1.

Examples:

```
s = "leetcode"
return 0.

s = "loveleetcode",
return 2.
```

Note: You may assume the string contain only lowercase letters.

Lesson Learned:

`collections.Counter(s)` is ***extremely fast*** (more than dict). Behave similar to:

```
d={}
for c in s:
    d[c]=d.get(c,0)+1
```

392. Is Subsequence

Given a string **s** and a string **t**, check if **s** is subsequence of **t**.

You may assume that there is only lower case English letters in both **s** and **t**. **t** is potentially a very long (length ~= 500,000) string, and **s** is a short string (<=100).

A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, `"ace"` is a subsequence of `"abcde"` while `"aec"` is not).

Example 1:
s = `"abc"` , **t** = `"ahbgdc"`

Return `true` .

Example 2:
s = `"axc"` , **t** = `"ahbgdc"`

Return `false` .

Follow up:
If there are lots of incoming S, say S1, S2, ... , Sk where k >= 1B, and you want to check one by one to see if T has its subsequence. In this scenario, how would you change your code?

Credits:
Special thanks to @pbrother (https://leetcode.com/pbrother/) for adding this problem and creating all test cases.

how to loop thru string fast:

```
class Solution:
    # use index and in which is very fast (40 ms)
    def isSubsequence(self, s: str, t: str) -> bool:
        if not s: return True
        if not t: return False

        if s[0] in t:
             return self.isSubsequence(s[1:],t[t.index(s[0])+1:])
        return False
    # use iteration (64 ms)
    def isSubsequence2(self, s: str, t: str) -> bool:
        t=iter(t)
        return all(c in t for c in s)
    # increment very slow (288 ms)
    def isSubsequence1(self, s: str, t: str) -> bool:
        i=j=0
        while j < len(t):
            if i == len(s): break
            if s[i] == t[j]:
                i+=1
            j+=1
        return i == len(s)
```

404. Sum of Left Leaves

Find the sum of all left leaves in a given binary tree.

Example:

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

There are two left leaves in the binary tree, with values 9 and 15 respectively. Return 24.
```

Seperate left and right using flags

```
class Solution:
    # recursive
    def sumOfLeftLeaves(self, node: TreeNode,left=None) -> int:
        if not node: return 0
        if node.left is node.right is None:
            return node.val if left else 0
        return self.sumOfLeftLeaves(node.left,1) + self.sumOfLeftLeaves(node.right,0)
    # Iterative
    def sumOfLeftLeaves(self, root: TreeNode) -> int:
        result = 0
        stack = [(root, False)]
        while stack:
            curr, is_left = stack.pop()
            if not curr:
                continue
            if not curr.left and not curr.right:
                if is_left:
                    result += curr.val
            else:
                stack.append((curr.left, True))
                stack.append((curr.right, False))
        return result
```

414. Third Maximum Number

Given a **non-empty** array of integers, return the **third** maximum number in this array. If it does not exist, return the maximum number. The time complexity must be in O(n).

Example 1:

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

Output: 1

Explanation: The third maximum is 1.
```

Example 2:

```
Input: [1, 2]

Output: 2

Explanation: The third maximum does not exist, so the maximum (2) is returned instead.
```

Example 3:

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

Output: 1

Explanation: Note that the third maximum here means the third maximum distinct number.
Both numbers with value 2 are both considered as second maximum.
```

1st (bottom): is pythonic 2nd (top): is linear

```
class Solution:
    # 60ms, 85%
    def thirdMax(self, nums: List[int]) -> int:
        ans = [float("-inf")]*3
        for num in set(nums):
            if num > ans[0]:
                ans[0], ans[1], ans[2] = num, ans[0], ans[1]
            elif num > ans[1]:
                ans[1], ans[2] = num, ans[1]
            elif num > ans[2]:
                ans[2] = num
        return ans[0] if ans[-1] == float("-inf") else ans[-1]
    # 60ms, 85%
    def thirdMax1(self, nums: List[int]) -> int:
        nums = sorted(set(nums))
        return nums[-1] if len(nums)<=2 else nums[-3]
```

415. Add Strings

Given two non-negative integers `num1` and `num2` represented as string, return the sum of `num1` and `num2`.

Note:

1. The length of both `num1` and `num2` is < 5100.
2. Both `num1` and `num2` contains only digits `0-9`.
3. Both `num1` and `num2` does not contain any leading zero.
4. You **must not use any built-in BigInteger library** or **convert the inputs to integer** directly.

Elegant nested recursive call

```
class Solution:
    def addStrings(self, num1: str, num2: str) -> str:
        if num1 == "": return num2
        if num2 == "": return num1
        sm=int(num1[-1])+int(num2[-1])
        if sm >= 10:  # if sum>=10, the carry will be an argument for a nested recursive call
            return self.addStrings( self.addStrings( num1[:-1], num2[:-1] ), "1") + str( sm -
10 )
        # sm < 10
        return self.addStrings(num1[:-1],num2[:-1])+str(sm)
```

434. Number of Segments in a String

Count the number of segments in a string, where a segment is defined to be a contiguous sequence of non-space characters.

Please note that the string does not contain any **non-printable** characters.

Example:

```
Input: "Hello, my name is John"
Output: 5
```

Lession learn:

for problem that required counting words like `__space__word__space__word__space__word` . Pay attention to this:

```
counter=0
        for i in range(len(s)):
            if (i==0 or s[i-1]==" ") and s[i]!=" ":
                counter+=1
        return counter
```

442. Find All Duplicates in an Array

Given an array of integers, 1 ≤ a[i] ≤ *n* (*n* = size of array), some elements appear **twice** and others appear **once**.

Find all the elements that appear **twice** in this array.

Could you do it without extra space and in O(*n*) runtime?

Example:

```
Input:
[4,3,2,7,8,2,3,1]

Output:
[2,3]
```

Learn: when list with len = N contains number from 0 to N-1 or 1 to N. We can use the num as index to check for repetition.

```
class Solution:
    # 436 ms, 40%
    def findDuplicates(self, nums: List[int]) -> List[int]:
        ans = []
        for num in nums:
            if nums[abs(num)-1] < 0:
                ans.append(abs(num))
            else:
                nums[abs(num)-1] = -nums[abs(num)-1]
        return ans

    # 384ms, 99%
    def findDuplicates1(self, nums: List[int]) -> List[int]:
        L = [0]*(len(nums)+1)
        ans = []
        for num in nums:
            if L[num]: ans.append(num)
            else: L[num] = 1
        return ans
```

448. Find All Numbers Disappeared in an Array

Given an array of integers where 1 ≤ a[i] ≤ *n* (*n* = size of array), some elements appear twice and others appear once.

Find all the elements of [1, *n*] inclusive that do not appear in this array.

Could you do it without extra space and in O(*n*) runtime? You may assume the returned list does not count as extra space.

Example:

```
Input:
[4,3,2,7,8,2,3,1]

Output:
[5,6]
```

initial sol: use `set()` However, the problem requires no extra space

```
    def findDisappearedNumbers(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]
        """
        # For each number i in nums,
        # we mark the number that i points as negative.
        # Then we filter the list, get all the indexes
        # who points to a positive number
        for i in xrange(len(nums)):
            index = abs(nums[i]) - 1
            nums[index] = - abs(nums[index])

        return [i + 1 for i in range(len(nums)) if nums[i] > 0]

    # using set
    def findDisappearedNumbers(self, nums: List[int]) -> List[int]:
        return set(range(1, len(nums)+1)) - set(nums)
```

459. Repeated Substring Pattern

Given a non-empty string check if it can be constructed by taking a substring of it and appending multiple copies of the substring together. You may assume the given string consists of lowercase English letters only and its length will not exceed 10000.

Example 1:

```
Input: "abab"
Output: True
Explanation: It's the substring "ab" twice.
```

Example 2:

```
Input: "aba"
Output: False
```

Example 3:

```
Input: "abcabcabcabc"
Output: True
Explanation: It's the substring "abc" four times. (And the substring "abcabc" twice.)
```

Lesson Learned:

1. Post (https://leetcode.com/problems/repeated-substring-pattern/discuss/94334/Easy-python-solution-with-explaination)
2. If a string equal to some rotation of itself -> it's periodic
3. Difference between: `2*s[1:-1]` : subscripting before multiply `(2*s)[1:-1]` multiply before subscripting

468. Validate IP Address

Write a function to check whether an input string is a valid IPv4 address or IPv6 address or neither.

IPv4 addresses are canonically represented in dot-decimal notation, which consists of four decimal numbers, each ranging from 0 to 255, separated by dots ("."), e.g., `172.16.254.1`;

Besides, leading zeros in the IPv4 is invalid. For example, the address `172.16.254.01` is invalid.

IPv6 addresses are represented as eight groups of four hexadecimal digits, each group representing 16 bits. The groups are separated by colons (":"). For example, the address `2001:0db8:85a3:0000:0000:8a2e:0370:7334` is a valid one. Also, we could omit some leading zeros among four hexadecimal digits and some low-case characters in the address to upper-case ones, so `2001:db8:85a3:0:0:8A2E:0370:7334` is also a valid IPv6 address(Omit leading zeros and using upper cases).

However, we don't replace a consecutive group of zero value with a single empty group using two consecutive colons (::) to pursue simplicity. For example, `2001:0db8:85a3::8A2E:0370:7334` is an invalid IPv6 address.

Besides, extra leading zeros in the IPv6 is also invalid. For example, the address `02001:0db8:85a3:0000:0000:8a2e:0370:7334` is invalid.

Note: You may assume there is no extra space or special characters in the input string.

Example 1:

```
Input: "172.16.254.1"

Output: "IPv4"

Explanation: This is a valid IPv4 address, return "IPv4".
```

Example 2:

```
Input: "2001:0db8:85a3:0:0:8A2E:0370:7334"

Output: "IPv6"

Explanation: This is a valid IPv6 address, return "IPv6".
```

Example 3:

```
Input: "256.256.256.256"

Output: "Neither"

Explanation: This is neither a IPv4 address nor a IPv6 address.
```

Lesson learned:

2 ways to check IP4, IP 6

1.String manipulation

```
class Solution:
    def validIPAddress(self, IP: str) -> str:
        def IP4(num):
            try: return str(int(num))==num and 0<=int(num)<256
            except: return 0

        def IP6(num):
            if len(num)>4 or len(num)==0: return 0
            try: return all([1 if c in "abcdefABCDEF0123456789" else 0 for c in num ])
            except: return 0

        if IP.count(".")==3 and all(IP4(i) for i in IP.split(".")):
            return "IPv4"
        elif IP.count(":")==7 and all(IP6(i) for i in IP.split(":")):
            return "IPv6"
        return "Neither"
```

2.Regex

```
class Solution:
    def validIPAddress2(self, IP: str) -> str:
        import re
        match=re.match(r"^(((1?[1-9]?\d)|(2[0-4]\d)|(25[0-5]))\.){3}((1?[1-9]?\d)|(2[0-4]\
d)|(25[0-5]))$",IP)
        if match: return "IPv4"

        match=re.match(r"^([a-fA-F0-9]{1,4}:){7}[a-fA-F0-9]{1,4}$",IP)
        if match: return "IPv6"

        return "Neither"
```

482. License Key Formatting

You are given a license key represented as a string S which consists only alphanumeric character and dashes. The string is separated into N+1 groups by N dashes.

Given a number K, we would want to reformat the strings such that each group contains *exactly* K characters, except for the first group which could be shorter than K, but still must contain at least one character. Furthermore, there must be a dash inserted between two groups and all lowercase letters should be converted to uppercase.

Given a non-empty string S and a number K, format the string according to the rules described above.

Example 1:

```
Input: S = "5F3Z-2e-9-w", K = 4

Output: "5F3Z-2E9W"

Explanation: The string S has been split into two parts, each part has 4 characters.
Note that the two extra dashes are not needed and can be removed.
```

Example 2:

```
Input: S = "2-5g-3-J", K = 2

Output: "2-5G-3J"

Explanation: The string S has been split into three parts, each part has 2 characters except th
```

Note:

1. The length of string S will not exceed 12,000, and K is a positive integer.
2. String S consists only of alphanumerical characters (a-z and/or A-Z and/or 0-9) and dashes(-).
3. String S is non-empty.

Self post (https://leetcode.com/problems/license-key-formatting/discuss/352717/Python-3-short-simple-solution-with-explanation-(-and-fast))

513. Find Bottom Left Tree Value

Given a binary tree, find the leftmost value in the last row of the tree.

Example 1:

```
Input:

    2
   / \
  1   3

Output:
1
```

Example 2:

```
Input:

        1
       / \
      2   3
     /   / \
    4   5   6
       /
      7

Output:
7
```

Note: You may assume the tree (i.e., the given root node) is not **NULL**.

```
    # very creative, we do right to left bfs
    def findBottomLeftValue(self, root: TreeNode) -> int:
        q = [root]
        for node in q:
            if node.right: q.append(node.right)
            if node.left: q.append(node.left)
        return node.val
```

520. Detect Capital

Given a word, you need to judge whether the usage of capitals in it is right or not.

We define the usage of capitals in a word to be right when one of the following cases holds:

1. All letters in this word are capitals, like "USA".
2. All letters in this word are not capitals, like "leetcode".
3. Only the first letter in this word is capital, like "Google".

Otherwise, we define that this word doesn't use capitals in a right way.

Example 1:

```
Input: "USA"
Output: True
```

Example 2:

```
Input: "FlaG"
Output: False
```

Note: The input will be a non-empty word consisting of uppercase and lowercase latin letters.

```
Can check Capital by just checking s[1:].islower()
```

521. Longest Uncommon Subsequence I

Given a group of two strings, you need to find the longest uncommon subsequence of this group of two strings. The longest uncommon subsequence is defined as the longest subsequence of one of these strings and this subsequence should not be **any** subsequence of the other strings.

A **subsequence** is a sequence that can be derived from one sequence by deleting some characters without changing the order of the remaining elements. Trivially, any string is a subsequence of itself and an empty string is a subsequence of any string.

The input will be two strings, and the output needs to be the length of the longest uncommon subsequence. If the longest uncommon subsequence doesn't exist, return -1.

Example 1:

```
Input: "aba", "cdc"
Output: 3
Explanation: The longest uncommon subsequence is "aba" (or "cdc"),
because "aba" is a subsequence of "aba",
but not a subsequence of any other strings in the group of two strings.
```

Note:

1. Both strings' lengths will not exceed 100.
2. Only letters from a ~ z will appear in input strings.

Can use float to check validity

530. Minimum Absolute Difference in BST

Given a binary search tree with non-negative values, find the minimum absolute difference (https://en.wikipedia.org/wiki/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
```

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

Same as #783 (https://leetcode.com/problems/minimum-distance-between-bst-nodes/description/)

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def getMinimumDifference(self, node: TreeNode,lo=float("-inf"),hi=float("inf")) -> int:
        if node:
            left=self.getMinimumDifference(node.left,lo,node.val)
            right=self.getMinimumDifference(node.right,node.val,hi)
            return min(left,right)
        return hi-lo
```

541. Reverse String II

Given a string and an integer k, you need to reverse the first k characters for every 2k characters counting from the start of the string. If there are less than k characters left, reverse all of them. If there are less than 2k but greater than or equal to k characters, then reverse the first k characters and left the other as original.

Example:

```
Input: s = "abcdefg", k = 2
Output: "bacdfeg"
```

Restrictions:

1. The string consists of lower English letters only.

2. Length of the given string and k will in the range [1, 10000]

Lesson Learned:

1. `reversed(lst)`

565. Array Nesting

A zero-indexed array A of length N contains all integers from 0 to N-1. Find and return the longest length of set S, where S[i] = {A[i], A[A[i]], A[A[A[i]]], ... } subjected to the rule below.

Suppose the first element in S starts with the selection of element A[i] of index = i, the next element in S should be A[A[i]], and then A[A[A[i]]]… By that analogy, we stop adding right before a duplicate element occurs in S.

Example 1:

```
Input: A = [5,4,0,3,1,6,2]
Output: 4
Explanation:
A[0] = 5, A[1] = 4, A[2] = 0, A[3] = 3, A[4] = 1, A[5] = 6, A[6] = 2.

One of the longest S[K]:
S[0] = {A[0], A[5], A[6], A[2]} = {5, 6, 2, 0}
```

Note:

1. N is an integer within the range [1, 20,000].
2. The elements of A are all distinct.
3. Each element of A is an integer within the range [0, N-1].

2 solutions by me

```
class Solution:
    # approved by in-place memoization
    # so we don't have to check i in dct and nums[i] in s
    # Space complexity O(1)
    # 132 ms, 77%
    def arrayNesting(self, nums: List[int]) -> int:
        mx = 0
        for i in range(len(nums)):
            if nums[i] == -1: continue # continue if we already met it
            length = 0
            while nums[i] != -1: # stop when we meet the same number twice
                nums[i], i = -1, nums[i]
                length += 1
            if length > mx: mx = length
        return mx

    # initial approach: using set, dictionary
    # 136 ms, 56%
    def arrayNesting(self, nums: List[int]) -> int:
        dct = {}
        mx = 0
        for i in range(len(nums)):
            if i in dct: continue
            s = set()
            while nums[i] not in s:
                s.add(nums[i])
                i = nums[i]
            for num in s:
                dct[num] = len(s)
            mx = max(mx, len(s))
        return mx
```

566. Reshape the Matrix

In MATLAB, there is a very useful function called 'reshape', which can reshape a matrix into a new one with different size but keep its original data.

You're given a matrix represented by a two-dimensional array, and two **positive** integers **r** and **c** representing the **row** number and **column** number of the wanted reshaped matrix, respectively.

The reshaped matrix need to be filled with all the elements of the original matrix in the same **row-traversing** order as they were.

If the 'reshape' operation with given parameters is possible and legal, output the new reshaped matrix; Otherwise, output the original matrix.

Example 1:

```
Input:
nums =
[[1,2],
 [3,4]]
r = 1, c = 4
Output:
[[1,2,3,4]]
Explanation:
The row-traversing of nums is [1,2,3,4]. The new reshaped matrix is a 1 * 4 matrix, fill it row
```

Example 2:

```
Input:
nums =
[[1,2],
 [3,4]]
r = 2, c = 4
Output:
[[1,2],
 [3,4]]
Explanation:
There is no way to reshape a 2 * 2 matrix to a 2 * 4 matrix. So output the original matrix.
```

Note:

1. The height and width of the given matrix is in range [1, 100].
2. The given r and c are all positive.

Many different sol by Stefan post (https://leetcode.com/problems/reshape-the-matrix/discuss/102500/Python-Solutions)

Learn about:

- `itertools.chain()` noted that this return a itertools, so if we can only traverse thru it once.
- `itertools.islice()` this is `slice()` applied to an `iterator`. It has to be an iterator, else it will slice the same part over and over again.
- `sum([[1, 2], [3, 4]], [])` will concatenate the list because the `start` is `[]` which means `[] + [1, 2] + [3, 4]`
-

605. Can Place Flowers

Suppose you have a long flowerbed in which some of the plots are planted and some are not. However, flowers cannot be planted in adjacent plots - they would compete for water and both would die.

Given a flowerbed (represented as an array containing 0 and 1, where 0 means empty and 1 means not empty), and a number **n**, return if **n** new flowers can be planted in it without violating the no-adjacent-flowers rule.

Example 1:

```
Input: flowerbed = [1,0,0,0,1], n = 1
Output: True
```

Example 2:

```
Input: flowerbed = [1,0,0,0,1], n = 2
Output: False
```

Note:

1. The input array won't violate no-adjacent-flowers rule.
2. The input array size is in the range of [1, 20000].
3. **n** is a non-negative integer which won't exceed the input array size.

Clean code or high performance code?

```
class Solution:
    # 196 ms 48%
    def canPlaceFlowers(self, flowerbed: List[int], n: int) -> bool:
        flowerbed = [0] + flowerbed + [0]
        for i in range(1, len(flowerbed)-1):
            if 1 not in flowerbed[i-1 : i+2]:
                flowerbed[i] = 1
                n -= 1
        return n <= 0
    # 184 ms, 92%
    def canPlaceFlowers1(self, flowerbed: List[int], n: int) -> bool:
        i=0
        while i < len(flowerbed):
            if flowerbed[i]:
                i += 2
                continue
            if i == len(flowerbed)-1 or not flowerbed[i + 1]:
                n -= 1
                i += 2
            else:
                i += 1
        return n <= 0
```

606. Construct String from Binary Tree

You need to construct a string consists of parenthesis and integers from a binary tree with the preorder traversing way.

The null node needs to be represented by empty parenthesis pair "()". And you need to omit all the empty parenthesis pairs that don't affect the one-to-one mapping relationship between the string and the original binary tree.

Example 1:

```
Input: Binary tree: [1,2,3,4]
       1
     /   \
    2     3
   /
  4

Output: "1(2(4))(3)"

Explanation: Originallay it needs to be "1(2(4)())(3()())",
but you need to omit all the unnecessary empty parenthesis pairs.
And it will be "1(2(4))(3)".
```

Example 2:

```
Input: Binary tree: [1,2,3,null,4]
       1
     /   \
    2     3
     \
      4

Output: "1(2()(4))(3)"

Explanation: Almost the same as the first example,
except we can't omit the first parenthesis pair to break the one-to-one mapping relationship be
```

Lesson Learned:

Binary Tree

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def tree2str(self, t: TreeNode) -> str:

        def disp(node):
            if node==None:
                return ""

            if node.left==None and node.right==None:
                return str(node.val)
            if node.right==None:
                return "{}({})".format(node.val,disp(node.left))

            return "{}({})({})".format(node.val,disp(node.left),disp(node.right))
        return disp(t)
```

617. Merge Two Binary Trees

Given two binary trees and imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not.

You need to merge them into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of new tree.

Example 1:

```
Input:
        Tree 1                     Tree 2
          1                         2
         / \                       / \
        3   2                     1   3
       /                           \   \
      5                             4   7
Output:
Merged tree:
             3
            / \
           4   5
          / \   \
         5   4   7
```

Note: The merging process must start from the root nodes of both trees.

Self Improvement

first solution is longer but it can be improved for this case `if t1 is None or t2 is None` we can return the other tree instead of recursive combine it with None. Howevver it is not recommended to `return t1 or t2` because of memory management

It is also better to use t1 instead of creating a new Tree. This will be both memory efficient and faster.

```
class Solution:
    def mergeTrees(self, t1: TreeNode, t2: TreeNode) -> TreeNode:
        def combine(t1,t2):
            if t1==t2==None: return None
            if t1 is None or t2 is None: return t1 or t2
            t1.val+=t2.val
            t1.left=combine(t1.left,t2.left)
            t1.right=combine(t1.right,t2.right)
            return  t1
        t=combine(t1,t2)
        return t

    def mergeTrees1(self, t1: TreeNode, t2: TreeNode) -> TreeNode:
        def combine(t1,t2):
            if t1==t2==None: return None
            if t1 is None or t2 is None: return t1 or t2
            t=TreeNode(t1.val+t2.val)
            t.left=combine(t1.left,t2.left)
            t.right=combine(t1.right,t2.right)
            return  t
        t=combine(t1,t2)
        return t
    def mergeTrees1(self, t1: TreeNode, t2: TreeNode) -> TreeNode:
        def combine(t1,t2):
            if t1==t2==None: return None
            if not t2:
                t=TreeNode(t1.val)
                t.left=combine(t1.left,None)
                t.right=combine(t1.right,None)
                return t
            if not t1:
                t=TreeNode(t2.val)
                t.left=combine(None,t2.left)
                t.right=combine(None,t2.right)
                return t

            t=TreeNode(t1.val+t2.val)
            t.left=combine(t1.left,t2.left)
            t.right=combine(t1.right,t2.right)
            return  t
        t=combine(t1,t2)
        return t
```

661. Image Smoother

Given a 2D integer matrix M representing the gray scale of an image, you need to design a smoother to make the gray scale of each cell becomes the average gray scale (rounding down) of all the 8 surrounding cells and itself. If a cell has less than 8 surrounding cells, then use as many as you can.

Example 1:

```
Input:
[[1,1,1],
 [1,0,1],
 [1,1,1]]
Output:
[[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]]
Explanation:
For the point (0,0), (0,2), (2,0), (2,2): floor(3/4) = floor(0.75) = 0
For the point (0,1), (1,0), (1,2), (2,1): floor(5/6) = floor(0.83333333) = 0
For the point (1,1): floor(8/9) = floor(0.88888889) = 0
```

Note:

1. The value in the given matrix is in the range of [0, 255].
2. The length and width of the given matrix are in the range of [1, 150].

how to loop over 9 cells in a block: `for nr in (r-1, r, r+1): for nc in (c-1, c, c+1):`

```
class Solution:
    def imageSmoother(self, M: List[List[int]]) -> List[List[int]]:

        def new_value(r, c):
            s = count = 0
            for nr in (r-1, r, r+1):
                for nc in (c-1, c, c+1):
                    if 0 <= nr <= R-1 and 0 <= nc <= C-1:
                        s += M[nr][nc]
                        count += 1
            return s//count
        R, C = len(M), len(M[0])
        return [[new_value(r, c) for c in range(C) ] for r in range(R)]
```

665. Non-decreasing Array

Given an array with `n` integers, your task is to check if it could become non-decreasing by modifying **at most** `1` element.

We define an array is non-decreasing if `array[i] <= array[i + 1]` holds for every `i` (1 <= i < n).

Example 1:

```
Input: [4,2,3]
Output: True
Explanation: You could modify the first 4 to 1 to get a non-decreasing array.
```

Example 2:

```
Input: [4,2,1]
Output: False
Explanation: You can't get a non-decreasing array by modify at most one element.
```

Note: The `n` belongs to [1, 10,000].

Lesson learned:

1. We can check if list is sorted by `lst==sorted(lst)` as done in post (https://leetcode.com/problems/non-decreasing-array/discuss/106816/Python-Extremely-Easy-to-Understand)
2. my code

```
class Solution:
    def checkPossibility(self, nums: List[int]) -> bool:
        def is_increasing(arr):
            for i in range(len(arr)-1):
                if arr[i]>arr[i+1]:
                    return 0
            return 1

        for i in range(len(nums)-2):
            if nums[i]>nums[i+1]:
                if is_increasing( nums[:i+1]+nums[i+2:])\
                or is_increasing( nums[:i  ]+nums[i+1:]):
                    return True
                return False
        return True
```

671. Second Minimum Node In a Binary Tree

Given a non-empty special binary tree consisting of nodes with the non-negative value, where each node in this tree has exactly `two` or `zero` sub-node. If the node has two sub-nodes, then this node's value is the smaller value among its two sub-nodes. More formally, the property `root.val = min(root.left.val, root.right.val)` always holds.

Given such a binary tree, you need to output the **second minimum** value in the set made of all the nodes' value in the whole tree.

If no such second minimum value exists, output -1 instead.

Example 1:

```
Input:
    2
   / \
  2   5
     / \
    5   7

Output: 5
Explanation: The smallest value is 2, the second smallest value is 5.
```

Example 2:

```
Input:
    2
   / \
  2   2

Output: -1
Explanation: The smallest value is 2, but there isn't any second smallest value.
```

Example of BFS, iteration and recursive

```
class Solution:
    # 32 ms
    def findSecondMinimumValue(self, root: TreeNode) -> int:
        self.ans = float('inf')
        min1 = root.val
        def dfs(node):
            if node:
                if min1 < node.val < self.ans:
                    self.ans = node.val
                elif node.val == min1:
                    dfs(node.left)
                    dfs(node.right)
        dfs(root)
        return self.ans if self.ans < float('inf') else -1

    # veyr fast 28 ms
    def findSecondMinimumValue1(self, root: TreeNode) -> int:
        q=[root]
        min_diff=float("inf")
        for node in q:
            if node.left:
                if node.left.val==node.val:
                    q.append(node.left)
                else:
                    min_diff=min(node.left.val,min_diff)
                if node.right.val==node.val:
                    q.append(node.right)
                else:
                    min_diff=min(node.right.val,min_diff)
        return -1 if min_diff==float("inf") else min_diff
```

695. Max Area of Island

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.

Find the maximum area of an island in the given 2D array. (If there is no island, the maximum area is 0.)

Example 1:

```
[[0,0,1,0,0,0,0,1,0,0,0,0,0],
 [0,0,0,0,0,0,0,1,1,1,0,0,0],
 [0,1,1,0,1,0,0,0,0,0,0,0,0],
 [0,1,0,0,1,1,0,0,1,0,1,0,0],
 [0,1,0,0,1,1,0,0,1,1,1,0,0],
 [0,0,0,0,0,0,0,0,0,0,1,0,0],
 [0,0,0,0,0,0,0,1,1,1,0,0,0],
 [0,0,0,0,0,0,0,1,1,0,0,0,0]]
```

Given the above grid, return `6`. Note the answer is not 11, because the island must be connected

4-directionally.

Example 2:

```
[[0,0,0,0,0,0,0,0]]
```

Given the above grid, return `0`.

Note: The length of each dimension in the given `grid` does not exceed 50.

Lesson Learned:

DSF and BSF are pretty simple

```
class Solution:
        # My BFS. 156ms
    def maxAreaOfIsland(self, grid: List[List[int]]) -> int:
        R,C=len(grid),len(grid[0])

        from collections import deque
        def bsf(r,c):
            d=deque()
            d.append((r,c))
            s=0
            while(d):
                r,c=d.popleft()
                if 0<=r<R and 0<=c<C and grid[r][c]:
                    if grid[r][c]:
                        grid[r][c]=0
                        s+=1
                    d.append((r+1,c))
                    d.append((r-1,c))
                    d.append((r,c+1))
                    d.append((r,c-1))
            return s

        area=[bsf(r,c) for r in range(R) for c in range(C) if grid[r][c]]
        return max(area) if area else 0
```

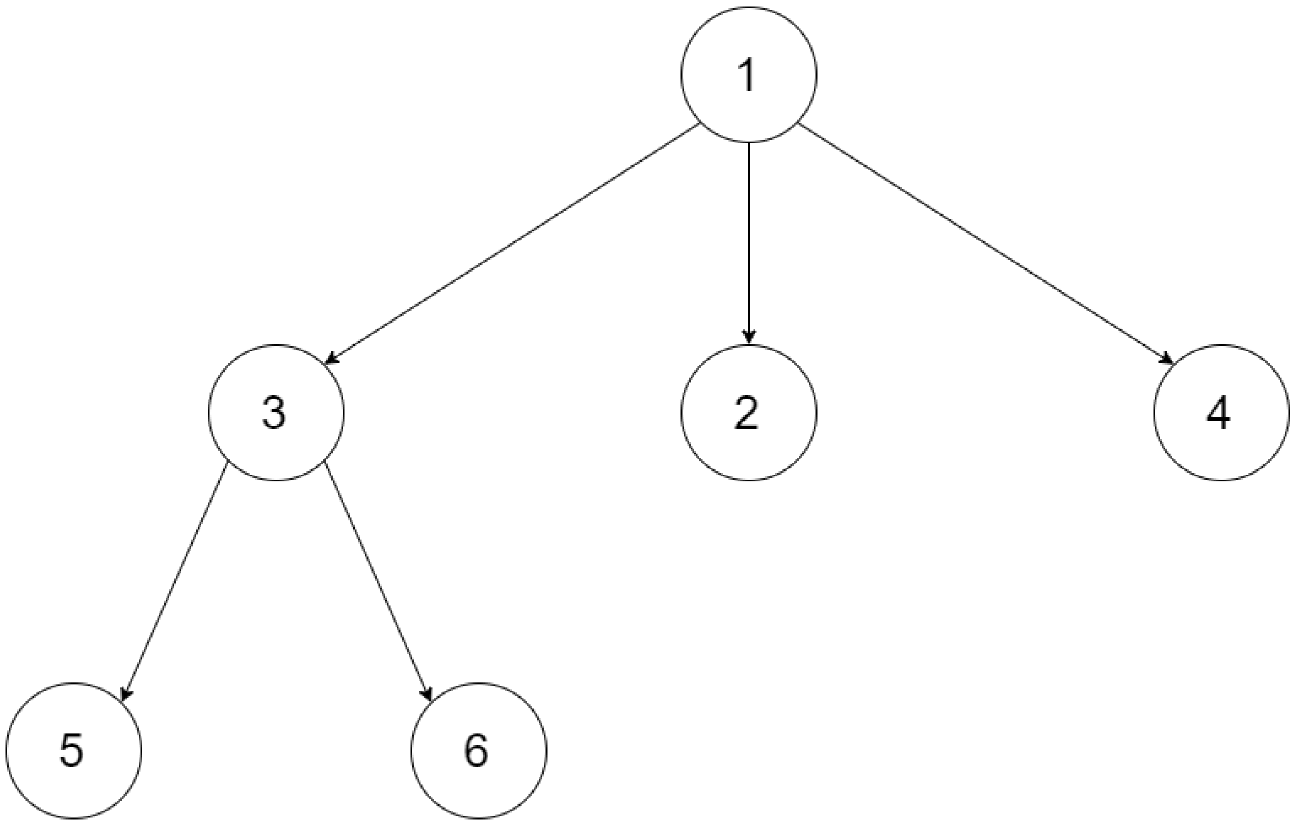
```
    # this is DSF with neat syntax. 148ms
    def maxAreaOfIsland3(self, grid: List[List[int]]) -> int:
        R,C=len(grid),len(grid[0])

        def dsf(r,c):
            if r<0 or c<0 or r>=R or c>=C\
                or grid[r][c]==0: return 0
            grid[r][c]=0 # Change current land to water
            return dsf(r+1,c) + dsf(r-1,c) + dsf(r,c+1) + dsf(r,c-1) +1 # +1 because of the l
and i am on.

        area=[dsf(r,c) for r in range(R) for c in range(C) if grid[r][c]]
        return max(area) if area else 0
        # My DFS. 148ms
    def maxAreaOfIsland2(self, grid: List[List[int]]) -> int:
        R,C=len(grid),len(grid[0])

        def mark_island(r,c):
            if r<0 or c<0 or r>=R or c>=C: return 0
            if grid[r][c]==0: return 0
            if grid[r][c]==1: grid[r][c]=0 # Change current land to water
            return mark_island(r+1,c) +  mark_island(r-1,c)\
                +  mark_island(r,c+1) +  mark_island(r,c-1) +1 # +1 because of the land i am
on.
        max_area=0
        for r in range(R):
            for c in range(C):
                if grid[r][c] == 1:
                    max_area=max(max_area,mark_island(r,c))
        return max_area
```

559. Maximum Depth of N-ary Tree

Given a n-ary 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.

For example, given a `3-ary` tree:



We should return its max depth, which is 3.

Note:

1. The depth of the tree is at most `1000` .
2. The total number of nodes is at most `5000` .

---

good post (https://leetcode.com/problems/maximum-depth-of-n-ary-tree/discuss/330666/3-Python-Solutions) that has 3 solutions: 2 line recursion, iteration BFS an DFS

2-line Recursion. In most tree problems, try recursion first.

```
class Solution(object):
    def maxDepth(self, root):
        if not root: return 0
        return 1 + max(map(self.maxDepth, root.children or [None]))
```

BFS (use a queue, the last level we see will be the depth)

```
class Solution(object):
    def maxDepth(self, root):
        queue = []
        if root: queue.append((root, 1))
        depth = 0
        for (node, level) in queue:
            depth = level
            queue += [(child, level+1) for child in node.children]
        return depth
```

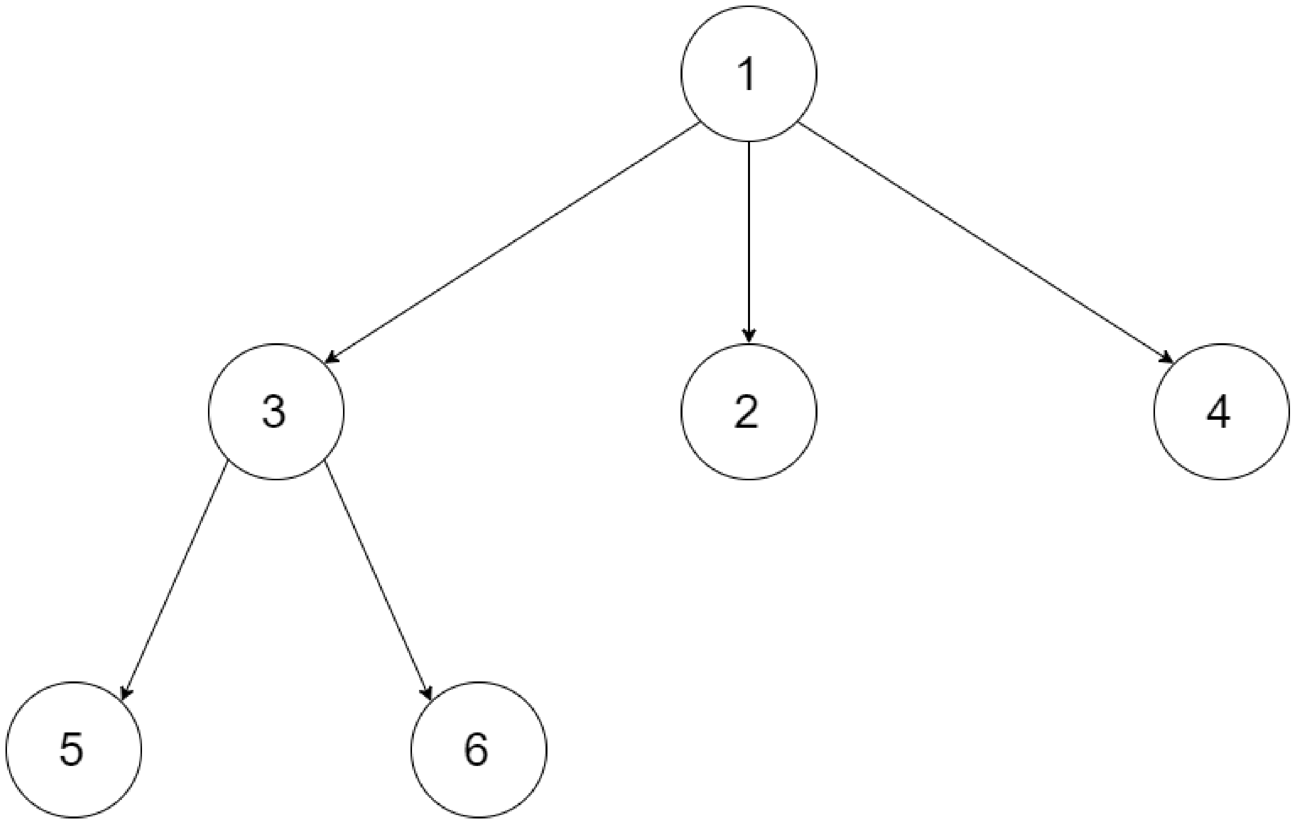
DFS (use a stack, use max to update depth)

```
class Solution(object):
    def maxDepth(self, root):
        stack = []
        if root: stack.append((root, 1))
        depth = 0
        while stack:
            (node, d) = stack.pop()
            depth = max(depth, d)
            for child in node.children:
                stack.append((child, d+1))
        return depth
```

589. N-ary Tree Preorder Traversal

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]` .

Note:

Recursive solution is trivial, could you do it iteratively?

Tree Preorder Traversal is the same as DFS. 2 ways

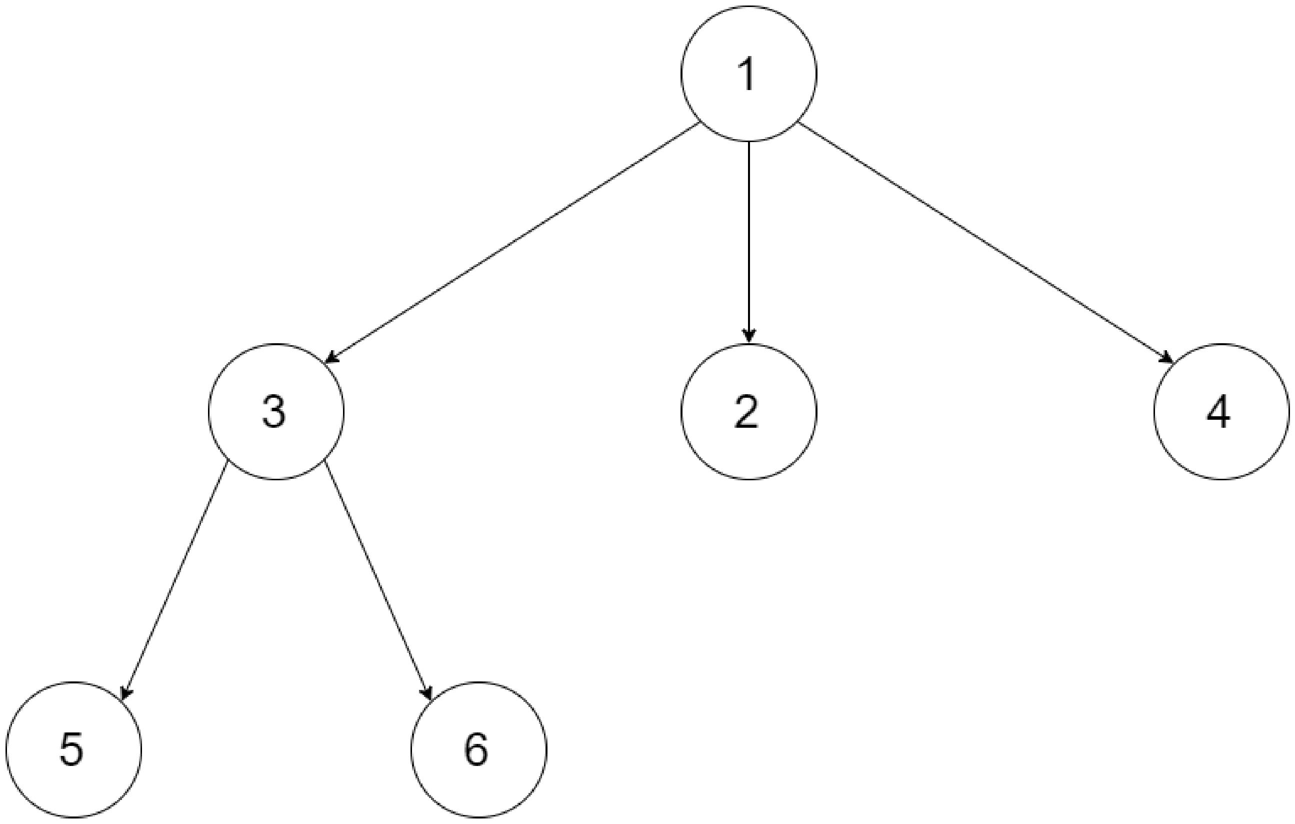
- iteration
- recursion

```
"""
# Definition for a Node.
class Node:
    def __init__(self, val, children):
        self.val = val
        self.children = children
"""
class Solution:
    def preorder(self, root: 'Node') -> List[int]:
        stack=[root]
        lst=[]
        while(stack):
            node=stack.pop()
            if node:
                lst.append(node.val)
                for child in reversed(node.children):
                    stack.append(child)
        return lst

    # recursive, slightlt different
    def preorder3(self, root: 'Node') -> List[int]:
        if not root: return
        rv=[root.val]
        for c in root.children:
            rv+=self.preorder(c)
        return rv

    # recursive
    def preorder(self, root: 'Node') -> List[int]:
        if not root: return []
        lst=[]
        def dfs(node):
            if node:
                lst.append(node.val)
                for child in node.children:
                    # or put this here instead of before for loop lst.append(child.val)
                    dfs(child)
        dfs(root)
        return lst
```

590. N-ary Tree Postorder Traversal

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?

---

2 ways. Iteration and recursion

```
# iteration
def postorder(self, node: 'Node') -> List[int]:
        if not node: return []
        stack=[node]
        lst=[]
        while(stack):
            node=stack.pop()
            lst.append(node.val)
            stack.extend(node.children)
        return reversed(lst)
# recursion
def postorder1(self, root: 'Node') -> List[int]:
        lst=[]
        if root:
                for c in root.children:
                        lst.extend(self.postorder(c))
                return lst+[root.val]
        return []
```

771. Jewels and Stones

You're given strings `J` representing the types of stones that are jewels, and `S` representing the stones you have. Each character in `S` is a type of stone you have. You want to know how many of the stones you have are also jewels.

The letters in `J` are guaranteed distinct, and all characters in `J` and `S` are letters. Letters are case sensitive, so `"a"` is considered a different type of stone from `"A"`.

Example 1:

```
Input: J = "aA", S = "aAAbbbb"
Output: 3
```

Example 2:

```
Input: J = "z", S = "ZZ"
Output: 0
```

Note:

- `S` and `J` will consist of letters and have length at most 50.
- The characters in `J` are distinct.

Lesson Learned:

there are many ways to count and sum

```
def numJewelsInStones(self, J: str, S: str) -> int:
        return sum(S.count(j) for j in J) # 32 - 56 ms
        return sum(c in J for c in S) # 36 ms
        return sum(map(S.count, J)) # 36 ms for c in J, count in S
        return sum(map(J.count, S)) # 40 ms for c in S, count in J
```

700. Search in a Binary Search Tree

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:
        4
       / \
      2   7
     / \
    1   3

And the value to search: 2
```

You should return this subtree:

```
      2
     / \
    1   3
```

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`.

2 ways:

- DFS. this problem use
- BFS

```
class Solution:

    # Recursive DFS
    def searchBST(self, node: TreeNode, val: int) -> TreeNode:
        def dfs(node):
            if node:
                if node.val == val:
                    return node
                if node.val > val:
                    return dfs(node.left)
                if node.val < val:
                    return dfs(node.right)
        return dfs(node)

    # Interative BFS
    def searchBST1(self, node: TreeNode, val: int) -> TreeNode:
        stack=[node]
        for node in stack:
            if node:
                if node.val==val:
                    return node
                elif node.val<val:
                    stack.append(node.right)
                elif node.val>val:
                    stack.append(node.left)
        return None
```

783. Minimum Distance Between BST Nodes

Given a Binary Search Tree (BST) with the root node `root` , return the minimum difference between the values of any two different nodes in the tree.

Example :

```
Input: root = [4,2,6,1,3,null,null]
Output: 1
Explanation:
Note that root is a TreeNode object, not an array.

The given tree [4,2,6,1,3,null,null] is represented by the following diagram:

          4
        /   \
      2      6
     / \
    1   3

while the minimum difference in this tree is 1, it occurs between node 1 and node 2, also betwe
```

Note:

1. The size of the BST will be between 2 and `100` .
2. The BST is always valid, each node's value is an integer, and each node's value is different.

Same as #530 (https://leetcode.com/problems/minimum-absolute-difference-in-bst/)

```
class Solution:
    # only calculate diff at the end (when node is None). So it's faster
    def minDiffInBST(self, root: TreeNode) -> int: # 36ms
        def dfs(node,low,high):
            if node:
                left=dfs(node.left,low,node.val)
                right=dfs(node.right,node.val,high)
                return min(left,right)
            return high-low
        return dfs(root,float("-inf"),float("inf"))

        # calculate diff at each node so it's slower
    def minDiffInBST(self, root: TreeNode) -> int: # 44ms
        def dfs(node,low,high):
            if node:
                m=float("inf")
                if node.left:
                    m=min(m,node.left.val-low, node.val-node.left.val)
                if node.right:
                    m=min(m,node.right.val-node.val,high-node.right.val)
                return min(m,dfs(node.left,low,node.val),dfs(node.right,node.val,high))
            return float("inf")
        return dfs(root,float("-inf"),float("inf"))
```

788. Rotated Digits

X is a good number if after rotating each digit individually by 180 degrees, we get a valid number that is different from X. Each digit must be rotated - we cannot choose to leave it alone.

A number is valid if each digit remains a digit after rotation. 0, 1, and 8 rotate to themselves; 2 and 5 rotate to each other; 6 and 9 rotate to each other, and the rest of the numbers do not rotate to any other number and become invalid.

Now given a positive number `N` , how many numbers X from `1` to `N` are good?

```
Example:
Input: 10
Output: 4
Explanation:
There are four good numbers in the range [1, 10] : 2, 5, 6, 9.
Note that 1 and 10 are not good numbers, since they remain unchanged after rotating.
```

Note:

- N will be in range `[1, 10000]`.

Good Solution

```
def rotatedDigits(self, N: int) -> int:
        # Fast method
        counter=0
        for n in range(N+1):
                n=str(n)
                if "3" in n or "4" in n or "7" in n: continue
                if "2" in n or "5" in n or "6" in n or "9" in n: counter+=1
        return counter
```

804. Unique Morse Code Words

International Morse Code defines a standard encoding where each letter is mapped to a series of dots and dashes, as follows: `"a"` maps to `".-"`, `"b"` maps to `"-..."`, `"c"` maps to `"-.-."`, and so on.

For convenience, the full table for the 26 letters of the English alphabet is given below:

```
[".-","-...","-.-.","-..",".","..-.","--.","....","..",".---","-.-",".-..","--","-.","---",".--
```

Now, given a list of words, each word can be written as a concatenation of the Morse code of each letter. For example, "cba" can be written as "-.-..--...", (which is the concatenation "-.-." + "-..." + ".-"). We'll call such a concatenation, the transformation of a word.

Return the number of different transformations among all words we have.

```
Example:
Input: words = ["gin", "zen", "gig", "msg"]
Output: 2
Explanation:
The transformation of each word is:
"gin" -> "--...-."
"zen" -> "--...-."
"gig" -> "--...--."
"msg" -> "--...--."

There are 2 different transformations, "--...-." and "--...--.".
```

Note:

- The length of `words` will be at most `100`.
- Each `words[i]` will have length in range `[1, 12]`.
- `words[i]` will only consist of lowercase letters.

Using dict and set to find unique transformations `string.ascii_lowercase` returns `abc...xyz`

```
class Solution:
    def uniqueMorseRepresentations(self, words: List[str]) -> int:
        code=[".-","-...","-.-.","-..",".","..-.","--.","....",".." \
             ,".---","-.-",".-..","--","-.","---",".--.","--.-" \
             ,".-.","...","-","..-","...-",".--","-..-","-.--","--.."]

        D=dict(zip(string.ascii_lowercase,code)) # string.ascii_lowercase returns "abc...xyz"

        s=set()
        for word in words:
            # add the joint of letter (in code format) to a set
            s.add("".join([D[letter] for letter in word]))
        return len(s)
```

819. Most Common Word

Given a paragraph and a list of banned words, return the most frequent word that is not in the list of banned words. It is guaranteed there is at least one word that isn't banned, and that the answer is unique.

Words in the list of banned words are given in lowercase, and free of punctuation. Words in the paragraph are not case sensitive. The answer is in lowercase.

Example:

```
Input:
paragraph = "Bob hit a ball, the hit BALL flew far after it was hit."
banned = ["hit"]
Output: "ball"
Explanation:
"hit" occurs 3 times, but it is a banned word.
"ball" occurs twice (and no other word does), so it is the most frequent non-banned word in the
Note that words in the paragraph are not case sensitive,
that punctuation is ignored (even if adjacent to words, such as "ball,"),
and that "hit" isn't the answer even though it occurs more because it is banned.
```

Note:

- `1 <= paragraph.length <= 1000`.
- `0 <= banned.length <= 100`.
- `1 <= banned[i].length <= 10`.
- The answer is unique, and written in lowercase (even if its occurrences in `paragraph` may have uppercase symbols, and even if it is a proper noun.)
- `paragraph` only consists of letters, spaces, or the punctuation symbols `!?',;.`
- There are no hyphens or hyphenated words.
- Words only consist of letters, never apostrophes or other punctuation symbols.

Lesson learn:

1. `string.replace()` is not inplace
2. dictionary can be sorted with a key like this `sorted(d,key=d.get)`
3. max can be used with a key like this `max(d,key=d.get)`
4. `str.maketrans()` and `str.translate()`

```
removePunctuationTable = str.maketrans("!?',;.", "      ")
paragraph=paragraph.translate(removePunctuationTable)
```

832. Flipping an Image

Given a binary matrix `A`, we want to flip the image horizontally, then invert it, and return the resulting image.

To flip an image horizontally means that each row of the image is reversed. For example, flipping `[1, 1, 0]` horizontally results in `[0, 1, 1]`.

To invert an image means that each `0` is replaced by `1`, and each `1` is replaced by `0`. For example, inverting `[0, 1, 1]` results in `[1, 0, 0]`.

Example 1:

```
Input: [[1,1,0],[1,0,1],[0,0,0]]
Output: [[1,0,0],[0,1,0],[1,1,1]]
Explanation: First reverse each row: [[0,1,1],[1,0,1],[0,0,0]].
Then, invert the image: [[1,0,0],[0,1,0],[1,1,1]]
```

Example 2:

```
Input: [[1,1,0,0],[1,0,0,1],[0,1,1,1],[1,0,1,0]]
Output: [[1,1,0,0],[0,1,1,0],[0,0,0,1],[1,0,1,0]]
Explanation: First reverse each row: [[0,0,1,1],[1,0,0,1],[1,1,1,0],[0,1,0,1]].
Then invert the image: [[1,1,0,0],[0,1,1,0],[0,0,0,1],[1,0,1,0]]
```

Notes:

- `1 <= A.length = A[0].length <= 20`
- `0 <= A[i][j]` <= 1

Different variable of 1 liner

```
class Solution:
    def flipAndInvertImage(self, A: List[List[int]]) -> List[List[int]]:
        # This is my prefer way
        return [[ 0 if i else 1 for i in reversed(row)] for row in A]

        # We can index row instead of using reversed. Speed are similar
        return [[ 0 if i else 1 for i in row[::-1]] for row in A]

        # There are also different way to calculate value.
        # Using XOR
        return [[ i ^ 1 for i in reversed(row)] for row in A]

        # Or using int
        return [[ int(not i) for i in reversed(row)] for row in A]

        # Or simply subtraction
        return [[ 1 - i for i in reversed(row)] for row in A]
```

859. Buddy Strings

Given two strings `A` and `B` of lowercase letters, return `true` if and only if we can swap two letters in `A` so that the result equals `B`.

Example 1:

```
Input: A = "ab", B = "ba"
Output: true
```

Example 2:

```
Input: A = "ab", B = "ab"
Output: false
```

Example 3:

```
Input: A = "aa", B = "aa"
Output: true
```

Example 4:

```
Input: A = "aaaaaaabc", B = "aaaaaaacb"
Output: true
```

Example 5:

```
Input: A = "", B = "aa"
Output: false
```

Note:

1. `0 <= A.length <= 20000`
2. `0 <= B.length <= 20000`
3. `A` and `B` consist only of lowercase letters.

WIP

class Solution: def buddyStrings(self, A: str, B: str) -> bool: if len(A)!=len(B): return False for i in range(len(A)-1): if A[i]!=B[i]: return A[i+1]==B[i] and A[i]==B[i+1] and A[i+2:]==B[i+2:]

897. Increasing Order Search Tree

Given a binary search tree, rearrange the tree in **in-order** so that the leftmost node in the tree is now the root of the tree, and every node has no left child and only 1 right child.

```
Example 1:
Input: [5,3,6,2,4,null,8,1,null,null,null,7,9]

       5
      / \
    3    6
   / \    \
  2   4    8
 /        / \
1        7   9

Output: [1,null,2,null,3,null,4,null,5,null,6,null,7,null,8,null,9]

 1
  \
   2
    \
     3
      \
       4
        \
         5
          \
           6
            \
             7
              \
               8
                \
                 9
```

Note:

1. The number of nodes in the given tree will be between 1 and 100.
2. Each node will have a unique integer value from 0 to 1000.

Increasing order Tree 2 ways:

- inorder traversal
- dfs. However, in dfs, remember to make `node.left=None`

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    # dfs recursion
    def increasingBST(self, root):
        ans = self.cur = TreeNode(0)
        def inorder(node):
            if node:
                inorder(node.left)
                node.left = None
                self.cur.right = node
                self.cur = self.cur.right
                inorder(node.right)

        inorder(root)
        return ans.right

    # inorder traversal
    def increasingBST1(self, root: TreeNode) -> TreeNode:
        stack=[]
        rv=out=TreeNode(0)
        node=root
        while(stack or node):
            while(node):
                stack.append(node)
                node=node.left
            node=stack.pop()
            rv.right=TreeNode(node.val)
            rv=rv.right
            node=node.right
        return out.right
```

905. Sort Array By Parity

Given an array `A` of non-negative integers, return an array consisting of all the even elements of `A`, followed by all the odd elements of `A`.

You may return any answer array that satisfies this condition.

Example 1:

```
Input: [3,1,2,4]
Output: [2,4,3,1]
The outputs [4,2,3,1], [2,4,1,3], and [4,2,1,3] would also be accepted.
```

Note:

1. `1 <= A.length <= 5000`
2. `0 <= A[i] <= 5000`

Lesson Learned:

`e=o=[]` will map `e` and `o` to the same memory address meaning they are different name for the same list

```
class Solution:
    def sortArrayByParity0(self, A: List[int]) -> List[int]: #96 ms
        return sorted(A,key=lambda x: x%2==1)

    def sortArrayByParity(self, A: List[int]) -> List[int]: #100 ms
        o=[]
        e=[]
        for num in A:
            if num%2==0: e.append(num)
            else: o.append(num)
        return e+o

    def sortArrayByParity2(self, A: List[int]) -> List[int]: #88 ms
        left=0
        right=len(A)-1
        while(right>=left):
            if A[left]%2:
                while right>left and A[right]%2 :
                    right-=1
                A[right], A[left] = A[left], A[right]
            left+=1
        return A

    def sortArrayByParity1(self, A: List[int]) -> List[int]: #96ms
        left=0
        right=len(A)-1
        while(right>=left):
            if A[left] % 2 == 1 and A[right] % 2 == 0:
                A[right], A[left] = A[left], A[right]
                left+=1
                right-=1
            if A[left] % 2 == 0: left += 1
            if A[right] % 2 == 1: right -= 1
        return A
```

925. Long Pressed Name

Your friend is typing his `name` into a keyboard. Sometimes, when typing a character `c`, the key might get *long pressed*, and the character will be typed 1 or more times.

You examine the `typed` characters of the keyboard. Return `True` if it is possible that it was your friends name, with some characters (possibly none) being long pressed.

Example 1:

```
Input: name = "alex", typed = "aaleex"
Output: true
Explanation: 'a' and 'e' in 'alex' were long pressed.
```

Example 2:

```
Input: name = "saeed", typed = "ssaaedd"
Output: false
Explanation: 'e' must have been pressed twice, but it wasn't in the typed output.
```

Example 3:

```
Input: name = "leelee", typed = "lleeelee"
Output: true
```

Example 4:

```
Input: name = "laiden", typed = "laiden"
Output: true
Explanation: It's not necessary to long press any character.
```

Note:

1. `name.length <= 1000`
2. `typed.length <= 1000`
3. The characters of `name` and `typed` are lowercase letters.

Itertools.groupby() example

938. Range Sum of BST

Given the `root` node of a binary search tree, return the sum of values of all nodes with value between `L` and `R` (inclusive).

The binary search tree is guaranteed to have unique values.

Example 1:

```
Input: root = [10,5,15,3,7,null,18], L = 7, R = 15
Output: 32
```

Example 2:

```
Input: root = [10,5,15,3,7,13,18,1,null,6], L = 6, R = 10
Output: 23
```

Note:

1. The number of nodes in the tree is at most `10000`.
2. The final answer is guaranteed to be less than `2^31`.

Lesson Learned:

1. **Multiply** condition with return value instead of using `if condition`
2. Binary Property. left is less than right. We can use this to binary search the tree

```
class Solution:
    def rangeSumBST(self, root: TreeNode, L: int, R: int) -> int:
        if root is None: return 0
        if root.left==root.right==None:
            return root.val * (L<=root.val <=R)
        if root.val < L:
            return self.rangeSumBST(root.right, L, R)
        if root.val > R:
            return self.rangeSumBST(root.left, L, R)
        return root.val * (L<=root.val <=R)\
                + self.rangeSumBST(root.left, L, R)\
                + self.rangeSumBST(root.right, L, R)
```

941. Valid Mountain Array

Given an array `A` of integers, return `true` if and only if it is a *valid mountain array*.

Recall that A is a mountain array if and only if:

- `A.length >= 3`
- There exists some `i` with `0 < i < A.length - 1` such that:
    - `A[0] < A[1] < ... A[i-1] < A[i]`
    - `A[i] > A[i+1] > ... > A[A.length - 1]`

Example 1:

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

Example 2:

```
Input: [3,5,5]
Output: false
```

Example 3:

```
Input: [0,3,2,1]
Output: true
```

Note:

1. `0 <= A.length <= 10000`
2. `0 <= A[i] <= 10000`

Approach: Two people climb from left and from right separately. If they are climbing the same mountain, they will meet at the same point.

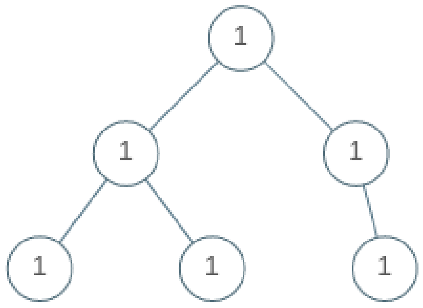
```
    def validMountainArray(self, A: List[int]) -> bool:
        if len(A) < 3: return False
        i = 0
        j = len(A)-1
        while i < len(A)-2 and A[i] < A[i+1]: i+=1
        while j > 1 and A[j] < A[j-1]: j-=1
        return i == j
```

965. Univalued Binary Tree

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

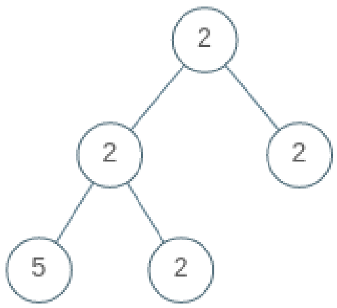
Return `true` if and only if the given tree is univalued.

Example 1:



```
Input: [1,1,1,1,1,null,1]
Output: true
```

Example 2:



```
Input: [2,2,2,5,2]
Output: false
```

Note:

1. The number of nodes in the given tree will be in the range `[1, 100]` .
2. Each node's value will be an integer in the range `[0, 99]` .

```
class Solution:
    # Iteration
    def isUnivalTree(self, root: TreeNode) -> bool:
        q=[root]
        for node in q:
            if node:
                q.extend([node.left,node.right])
                if node.left and node.left.val != node.val: return False
                if node.right and node.right.val != node.val: return False
        return True

    # Recursion
    def isUnivalTree1(self, root: TreeNode) -> bool:
        if not root: return True
        if root.left is root.right is None: return True
        if root.left and root.left.val != root.val: return False
        if root.right and root.right.val != root.val: return False
        return self.isUnivalTree(root.left) and self.isUnivalTree(root.right)
```

509. Fibonacci Number

The **Fibonacci numbers**, commonly denoted `F(n)` form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from `0` and `1` . That is,

```
F(0) = 0,   F(1) = 1
F(N) = F(N - 1) + F(N - 2), for N > 1.
```

Given `N` , calculate `F(N)` .

Example 1:

```
Input: 2
Output: 1
Explanation: F(2) = F(1) + F(0) = 1 + 0 = 1.
```

Example 2:

```
Input: 3
Output: 2
Explanation: F(3) = F(2) + F(1) = 1 + 1 = 2.
```

Example 3:

```
Input: 4
Output: 3
Explanation: F(4) = F(3) + F(2) = 2 + 1 = 3.
```

Note:

0 ≤ `N` ≤ 30.

Smart ways to calculate Fibonacci

1. `fib` using list. 28ms
2. `fib2` using 2 variables. 32 ms
3. `fib1` using dictionary. 36 ms

```
class Solution:
    def fib(self, N: int) -> int:
        lst=[0,1]
        if N<2: return N
        for i in range(1,N):
            lst.append(lst[-1]+lst[-2])
        return lst[-1]

    def fib2(self, N: int) -> int:
        a,b=0,1
        for i in range(N): a,b=b,a+b
        return a

    def fib1(self, N: int) -> int:
        known_fib={0:0,1:1}
        def get_fib(N):
            if N in known_fib:
                return known_fib[N]
            known_fib[N]=get_fib(N-1)+get_fib(N-2)
            return known_fib[N]
        return get_fib(N)
```

Other creative ways (https://leetcode.com/problems/fibonacci-number/discuss/308688/6-Python-solutions) including **golden ratio** and `from functools import lru_cache`

977. Squares of a Sorted Array

Given an array of integers `A` sorted in non-decreasing order, return an array of the squares of each number, also in sorted non-decreasing order.

Example 1:

```
Input: [-4,-1,0,3,10]
Output: [0,1,9,16,100]
```

Example 2:

```
Input: [-7,-3,2,3,11]
Output: [4,9,9,49,121]
```

Note:

1. `1 <= A.length <= 10000`
2. `-10000 <= A[i] <= 10000`
3. `A` is sorted in non-decreasing order.

Tons of different solution post (https://leetcode.com/problems/squares-of-a-sorted-array/discuss/310865/Python%3A-A-comparison-of-lots-of-approaches!-Sorting-two-pointers-deque-iterator-generator)

997. Find the Town Judge

In a town, there are `N` people labelled from `1` to `N`. There is a rumor that one of these people is secretly the town judge.

If the town judge exists, then:

1. The town judge trusts nobody.
2. Everybody (except for the town judge) trusts the town judge.
3. There is exactly one person that satisfies properties 1 and 2.

You are given `trust`, an array of pairs `trust[i] = [a, b]` representing that the person labelled `a` trusts the person labelled `b`.

If the town judge exists and can be identified, return the label of the town judge. Otherwise, return `-1`.

Example 1:

```
Input: N = 2, trust = [[1,2]]
Output: 2
```

Example 2:

```
Input: N = 3, trust = [[1,3],[2,3]]
Output: 3
```

Example 3:

```
Input: N = 3, trust = [[1,3],[2,3],[3,1]]
Output: -1
```

Example 4:

```
Input: N = 3, trust = [[1,2],[2,3]]
Output: -1
```

Example 5:

```
Input: N = 4, trust = [[1,3],[1,4],[2,3],[2,4],[4,3]]
Output: 3
```

Note:

1. `1 <= N <= 1000`
2. `trust.length <= 10000`
3. `trust[i]` are all different
4. `trust[i][0] != trust[i][1]`
5. `1 <= trust[i][0], trust[i][1] <= N`

Lesson learned:

Smart way to find judge:

```
class Solution:
    def findJudge(self, N: int, trust: List[List[int]]) -> int:
        d=[0]*(N+1)
        for i,j in trust:
            d[i]-=1
            d[j]+=1
        for i in range(1,N+1):
            if d[i]==N-1:
                return i
        return -1

    def findJudge2(self, N: int, trust: List[List[int]]) -> int:
        if len(trust)==0: return 1
        from collections import defaultdict
        dfd=defaultdict(set)
        s=set(range(1,N+1))
        for pair in trust:
            s.discard(pair[0])
            dfd[pair[1]].add(pair[0])
        # print(s)
        if len(s)==0: return -1
        for k,v in dfd.items():
            if len(v)==N-1:
                return k
        return -1
```

1180. Count Substrings with Only One Distinct Letter

Given a string `S`, return the number of substrings that have only **one distinct** letter.

Example 1:

```
Input: S = "aaaba"
Output: 8
Explanation: The substrings with one distinct letter are "aaa", "aa", "a", "b".
"aaa" occurs 1 time.
"aa" occurs 2 times.
"a" occurs 4 times.
"b" occurs 1 time.
So the answer is 1 + 2 + 4 + 1 = 8.
```

Example 2:

```
Input: S = "aaaaaaaaaa"
Output: 55
```

Constraints:

- `1 <= S.length <= 1000`
- `S[i]` consists of only lowercase English letters.

This problem was done in contest. It took me 45 minutes to do it because i didn't read the question carefully. However, the answer at the end is faily optimal.

Revised solution utilized `itertools.groupby)`

Lesson: need to read question carefully before doing it.

```
class Solution:

    # Revised, 28 ms
    def countLetters(self, S: str) -> int:
        import itertools
        total = 0
        for k, g in itertools.groupby(S):
            ln = len(list(g))
            total += (ln + 1)*ln/2
        return int(total)

    # Done in contest, 36 ms
    def countLetters(self, S: str) -> int:
        total = 0
        S += "-"
        start = 0
        for i in range(len(S)-1):
            if S[i] != S[i+1]:
                sub =S[start : i+1]
                total += (len(sub) + 1)*len(sub)/2
                start = i+1
        return int(total)
```

Another good solution. This is dynamic programming. Addition increases by 1 after each cycle if letter does not change. Answer is teh cummulation of the these additions.

```
#dynamic programming 36 ms
def countLetters(self, S: str) -> int:
        n = len(S)
        res = cur = 1
        for i in range(1, n):
                if S[i] == S[i - 1]:
                        cur += 1
                else:
                        cur = 1
                res += cur
        return res
```

1181. Before and After Puzzle

Given a list of `phrases` , generate a list of Before and After puzzles.

A *phrase* is a string that consists of lowercase English letters and spaces only. No space appears in the start or the end of a phrase. There are no consecutive spaces in a phrase.

Before and After puzzles are phrases that are formed by merging two phrases where the **last word of the first phrase** is the same as the **first word of the second phrase**.

Return the Before and After puzzles that can be formed by every two phrases `phrases[i]` and `phrases[j]` where `i != j` . Note that the order of matching two phrases matters, we want to consider both orders.

You should return a list of **distinct** strings **sorted lexicographically**.

Example 1:

```
Input: phrases = ["writing code","code rocks"]
Output: ["writing code rocks"]
```

Example 2:

```
Input: phrases = ["mission statement",
                  "a quick bite to eat",
                  "a chip off the old block",
                  "chocolate bar",
                  "mission impossible",
                  "a man on a mission",
                  "block party",
                  "eat my words",
                  "bar of soap"]
Output: ["a chip off the old block party",
         "a man on a mission impossible",
         "a man on a mission statement",
         "a quick bite to eat my words",
         "chocolate bar of soap"]
```

Example 3:

```
Input: phrases = ["a","b","a"]
Output: ["a"]
```

Constraints:

- `1 <= phrases.length <= 100`
- `1 <= phrases[i].length <= 100`

The solution done in contest is slower because `split()` is perform in every loop. We can improve this by only perform it once at the beginning.

```
class Solution:
    # pythonic, 40 ms
    def beforeAndAfterPuzzles(self, phrases: List[str]) -> List[str]:
        phrases = [phrase.split() for phrase in phrases]
        total = [phrases[i] + phrases[j][1:]
                for i in range(len(phrases))
                for j in range(len(phrases))
                if i != j and phrases[i][-1] == phrases[j][0]]
        return sorted(set([" ".join(phrase) for phrase in total]))

    # split() at the begining, 60 ms
    def beforeAndAfterPuzzles(self, phrases: List[str]) -> List[str]:
        phrases = [phrase.split() for phrase in phrases]
        total = []
        for i in range(len(phrases)):
            for j in range(len(phrases)):
                if i != j and phrases[i][-1] == phrases[j][0]:
                    total.append(phrases[i] + phrases[j][1:])
        return sorted(set([" ".join(phrase) for phrase in total]))

    # Done in contest, 196 ms
    def beforeAndAfterPuzzles(self, phrases: List[str]) -> List[str]:
        total = []
        for i in range(len(phrases)):
            for j in range(len(phrases)):
                phrase1 = phrases[i].split()
                phrase2 = phrases[j].split()
                if i != j and phrase1[-1] == phrase2[0]:
                     total.append(" ".join(phrase1 + phrase2[1:]))
        return sorted(set(total))
```

1182. Shortest Distance to Target Color

You are given an array `colors`, in which there are three colors: `1`, `2` and `3`.

You are also given some queries. Each query consists of two integers `i` and `c`, return the shortest distance between the given index `i` and the target color `c`. If there is no solution return `-1`.

Example 1:

```
Input: colors = [1,1,2,1,3,2,2,3,3], queries = [[1,3],[2,2],[6,1]]
Output: [3,0,3]
Explanation:
The nearest 3 from index 1 is at index 4 (3 steps away).
The nearest 2 from index 2 is at index 2 itself (0 steps away).
The nearest 1 from index 6 is at index 3 (3 steps away).
```

Example 2:

```
Input: colors = [1,2], queries = [[0,3]]
Output: [-1]
Explanation: There is no 3 in the array.
```

Constraints:

- `1 <= colors.length <= 5*10^4`
- `1 <= colors[i] <= 3`
- `1 <= queries.length <= 5*10^4`
- `queries[i].length == 2`
- `0 <= queries[i][0] < colors.length`
- `1 <= queries[i][1] <= 3`

Notice how this utilized `bisect()`

```
class Solution:
    # 2032 ms, using bisect
    def shortestDistanceColor(self, colors: List[int], queries: List[List[int]]) -> List[in
t]:
        D = collections.defaultdict(list)
        for i, v in enumerate(colors):
            D[v].append(i)
        print(D)
        ans = []
        for i, v in queries:
            if v not in D:
                ans.append(-1)
                continue
            index = bisect.bisect(D[v], i)
            temp1 = D[v][index] if index < len(D[v]) else float("inf")
            temp2 = D[v][index - 1] if index>0 else float("inf")
            ans.append(min(abs(i - temp1), abs(i - temp2)))
        return ans



    # done in contest 2100 ms, search left and right, naive
    def shortestDistanceColor1(self, colors: List[int], queries: List[List[int]]) -> List[in
t]:

        def nearest_left(index, val):
            for i in reversed(range(0, index +1)):
                if colors[i] == val:
                    return index - i
            return -1

        def nearest_right(index, val):
            for i in range(index, len(colors)):
                if colors[i] == val:
                    return i - index
            return -1

        ans = []
        D = {}
        for i, val in queries:
            if (i, val) in D:
                ans.append(D[(i,val)])
                continue
            left = nearest_left(i, val)
            right = nearest_right(i, val)
            if -1 in (left,right):
                rv = max(left, right)
            else:
                rv = min(left, right)
            ans.append(rv)
            D[(i,val)] = rv
        return ans
```

1183. Maximum Number of Ones

Consider a matrix `M` with dimensions `width * height`, such that every cell has value `0` or `1`, and any **square** sub-matrix of `M` of size `sideLength * sideLength` has at most `maxOnes` ones.

Return the maximum possible number of ones that the matrix `M` can have.

Example 1:

```
Input: width = 3, height = 3, sideLength = 2, maxOnes = 1
Output: 4
Explanation:
In a 3*3 matrix, no 2*2 sub-matrix can have more than 1 one.
The best solution that has 4 ones is:
[1,0,1]
[0,0,0]
[1,0,1]
```

Example 2:

```
Input: width = 3, height = 3, sideLength = 2, maxOnes = 2
Output: 6
Explanation:
[1,0,1]
[1,0,1]
[1,0,1]
```

Constraints:

- `1 <= width, height <= 100`
- `1 <= sideLength <= width, height`
- `0 <= maxOnes <= sideLength * sideLength`

```
class Solution:
    def maximumNumberOfOnes(self, C, R, K, maxOnes):
        # every K*K square has at most maxOnes ones
        count = [0] * (K*K)
        for r in range(R):
            for c in range(C):
                # calculate index to transform from 2D to 1D
                index = (r%K) * K + c%K
                count[index] += 1
        print(count)
        count.sort()
        ans = 0
        for _ in range(maxOnes):
            ans += count.pop()
        return ans
```

Consider a matrix M with dimensions width * height, such that every cell has value 0 or 1, and any square sub-matrix of M of size sideLength * sideLength has at most maxOnes ones.

Return the maximum possible number of ones that the matrix M can have.

Example 1:

```
Input: width = 3, height = 3, sideLength = 2, maxOnes = 1
Output: 4
Explanation:
In a 3*3 matrix, no 2*2 sub-matrix can have more than 1 one.
The best solution that has 4 ones is:
[1,0,1]
[0,0,0]
[1,0,1]
```

Example 2:

```
Input: width = 3, height = 3, sideLength = 2, maxOnes = 2
Output: 6
Explanation:
[1,0,1]
[1,0,1]
[1,0,1]
```

Constraints:

```
1 <= width, height <= 100
1 <= sideLength <= width, height
0 <= maxOnes <= sideLength * sideLength
```

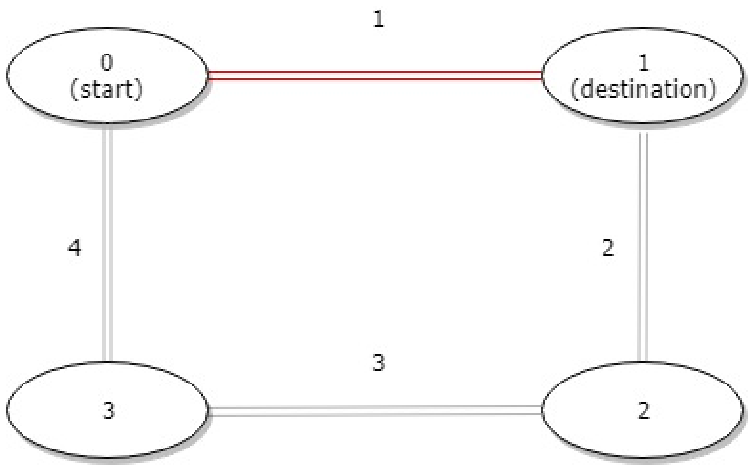
1184. Distance Between Bus Stops

A bus has `n` stops numbered from `0` to `n - 1` that form a circle. We know the distance between all pairs of neighboring stops where `distance[i]` is the distance between the stops number `i` and `(i + 1) % n`.

The bus goes along both directions i.e. clockwise and counterclockwise.

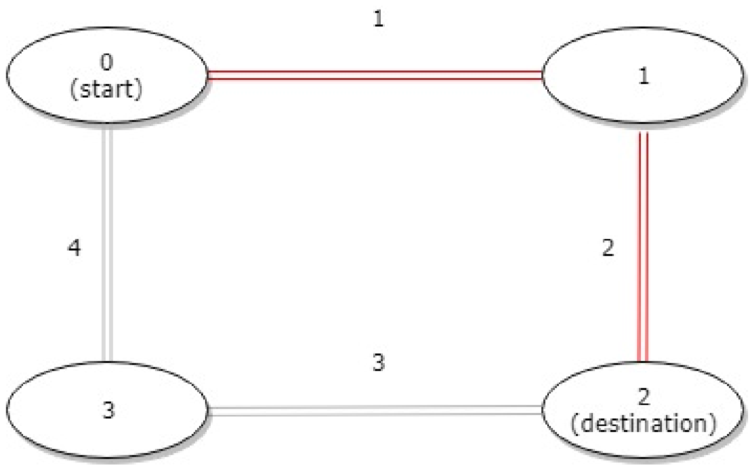
Return the shortest distance between the given `start` and `destination` stops.

Example 1:



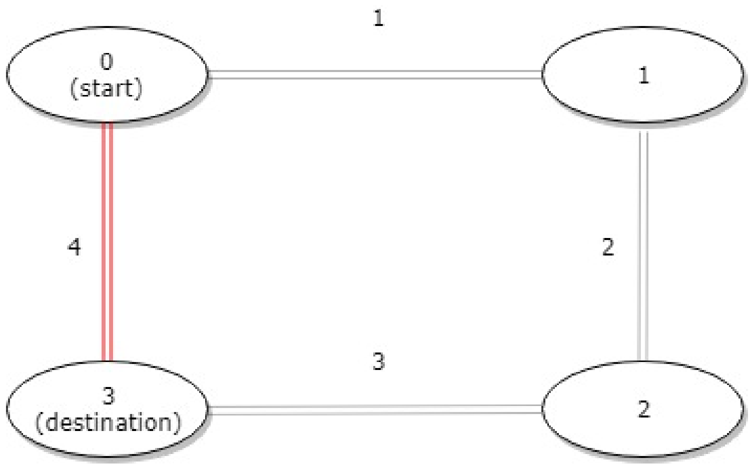
```
Input: distance = [1,2,3,4], start = 0, destination = 1
Output: 1
Explanation: Distance between 0 and 1 is 1 or 9, minimum is 1.
```

Example 2:



```
Input: distance = [1,2,3,4], start = 0, destination = 2
Output: 3
Explanation: Distance between 0 and 2 is 3 or 7, minimum is 3.
```

Example 3:



```
Input: distance = [1,2,3,4], start = 0, destination = 3
Output: 4
Explanation: Distance between 0 and 3 is 6 or 4, minimum is 4.
```

Constraints:

- `1 <= n <= 10^4`
- `distance.length == n`
- `0 <= start, destination < n`
- `0 <= distance[i] <= 10^4`

this problem is fairly easy, i did in within 10 mins

```
class Solution:
    # 56 ms, more pythonic
    def distanceBetweenBusStops(self, D: List[int], start: int, end: int) -> int:
        if start > end: start, end = end, start
        return min(sum(D[start : end]), sum(D[:start] + D[end:]))

    # 56 ms
    def distanceBetweenBusStops1(self, distance: List[int], start: int, destination: int) ->
int:
        if start > destination: start, destination = destination, start
        path1 = path2 = 0
        for i in range(len(distance)):
            if start <= i < destination:
                path1 += distance[i]
            else:
                path2 += distance[i]
        return min(path1, path2)
```

1185. Day of the Week

Given a date, return the corresponding day of the week for that date.
The input is given as three integers representing the `day`, `month` and `year` respectively.

Return the answer as one of the following values `{"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"}`.

Example 1:

```
Input: day = 31, month = 8, year = 2019
Output: "Saturday"
```

Example 2:

```
Input: day = 18, month = 7, year = 1999
Output: "Sunday"
```

Example 3:

```
Input: day = 15, month = 8, year = 1993
Output: "Sunday"
```

Constraints:

- The given dates are valid dates between the years `1971` and `2100`.

pay attention to how leap_year can be found: `return year%4 == 0 and (year%400 == 0 or year%100 != 0)`

```
class Solution:
    # built in, 32 ms
    def dayOfTheWeek(self, day: int, month: int, year: int) -> str:
        import datetime
        days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
        return days[datetime.datetime(year, month, day).weekday()]

    # calculation using math, 52ms
    def dayOfTheWeek(self, day: int, month: int, year: int) -> str:
        total_days = day

        def is_leap(year):
            return year%4 == 0 and (year%400 == 0 or year%100 != 0)

        for y in range(year):
            total_days += 366 if is_leap(y) else 365

        D = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

        leap_year = is_leap(year)
        for month in range(1, month):
            days = D[month-1] + leap_year if month == 2 else D[month-1]
            total_days += days

        D = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
        return D[(total_days + 4)%7]
```