Linked List Cycle

Given a linked list, determine if it has a cycle in it.

Follow up: Can you solve it without using extra space?

URL: https://leetcode.com/problems/linked-list-cycle/

```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def hasCycle(self, head):
        """
        :type head: ListNode
        :rtype: bool
        """
        if head == None:
            return False
        else:
            fast = head
            slow = head

            while fast != None and fast.next != None:
                slow = slow.next
                fast = fast.next.next
                if fast == slow:
                    break

            if fast == None or fast.next == None:
                return False
            elif fast == slow:
                return True

            return False
```

Reverse Linked List

Reverse a singly linked list.

URL: https://leetcode.com/problems/reverse-linked-list/

```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None


class Solution(object):
    def reverseList(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        if head == None:
            return None
        elif head != None and head.next == None:
            return head
        else:
            temp = None
            next_node = None
            while head != None:
                next_node = head.next
                head.next = temp
                temp = head
                head = next_node

            return temp
```

Delete Node in a Linked List

Write a function to delete a node (except the tail) in a singly linked list, given only access to that node.

Supposed the linked list is 1 -> 2 -> 3 -> 4 and you are given the third node with value 3, the linked list should become 1 -> 2 -> 4 after calling your function.

URL: https://leetcode.com/problems/delete-node-in-a-linked-list/

```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def deleteNode(self, node):
        """
        :type node: ListNode
        :rtype: void Do not return anything, modify node in-plac
e instead.
        """
        if node == None:
            pass
        else:
            next_node = node.next
            node.val = next_node.val
            node.next = next_node.next
```

Merge Two Sorted Lists

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

URL: https://leetcode.com/problems/merge-two-sorted-lists/

```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None


class Solution(object):
    def mergeTwoLists(self, l1, l2):
        """
        :type l1: ListNode
        :type l2: ListNode
        :rtype: ListNode
        """
        if l1 == None and l2 == None:
            return None
        elif l1 != None and l2 == None:
            return l1
        elif l2 != None and l1 == None:
            return l2
        else:
            dummy = ListNode(0)
            p = dummy

            while l1 != None and l2 != None:
                if l1.val < l2.val:
                    p.next = l1
                    l1 = l1.next
                else:
                    p.next = l2
                    l2 = l2.next
                p = p.next

            if l1 != None:
                p.next = l1

            if l2 != None:
                p.next = l2

            return dummy.next
```

Intersection of Two Linked Lists

Write a program to find the node at which the intersection of two singly linked lists begins.

For example, the following two linked lists:

A: a1 → a2 ↘ c1 → c2 → c3 ↗
B: b1 → b2 → b3 begin to intersect at node c1.

Notes:

If the two linked lists have no intersection at all, return null. The linked lists must retain their original structure after the function returns. You may assume there are no cycles anywhere in the entire linked structure. Your code should preferably run in O(n) time and use only O(1) memory.

URL: https://leetcode.com/problems/intersection-of-two-linked-lists/

```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def getIntersectionNode(self, headA, headB):
        """
        :type head1, head1: ListNode
        :rtype: ListNode
        """
        if headA == None and headB == None:
            return None
        elif headA == None and headB != None:
            return None
        elif headA != None and headB == None:
            return None
        else:
            len_a = 0
```

```
            len_b = 0

            current = headA
            while current != None:
                current = current.next
                len_a += 1

            current = headB
            while current != None:
                current = current.next
                len_b += 1

            diff = 0
            current = None
            if len_a > len_b:
                diff = len_a - len_b
                currentA = headA
                currentB = headB
            else:
                diff = len_b - len_a
                currentA = headB
                currentB = headA

            count = 0
            while count < diff:
                currentA = currentA.next
                count += 1

            while currentA != None and currentB != None:
                if currentA == currentB:
                    return currentA
                else:
                    currentA = currentA.next
                    currentB = currentB.next
```

Linked List Cycle II

Given a linked list, return the node where the cycle begins. If there is no cycle, return null.

Note: Do not modify the linked list.

Follow up: Can you solve it without using extra space?

URL: https://leetcode.com/problems/linked-list-cycle-ii/

```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def detectCycle(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        if head == None:
            return head
        else:
            fast = head
            slow = head

            has_cycle = False
            while fast != None and fast.next != None:
                slow = slow.next
                fast = fast.next.next
                if fast == slow:
                    has_cycle = True
                    break

            if has_cycle == False:
                return None

            slow = head
            while fast != slow:
                fast = fast.next
                slow = slow.next

            return slow
```

Palindrome Linked List

Given a singly linked list, determine if it is a palindrome.

Follow up: Could you do it in O(n) time and O(1) space?

URL: https://leetcode.com/problems/palindrome-linked-list/

```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def isPalindrome(self, head):
        """
        :type head: ListNode
        :rtype: bool
        """
        if head == None:
            return True
        elif head != None and head.next == None:
            return True
        else:
            fast = head
            slow = head
            stack = []
            while fast != None and fast.next != None:
                stack.append(slow.val)
                slow = slow.next
                fast = fast.next.next

            #madam
            if fast != None:
                slow = slow.next

            while slow != None:
                if slow.val != stack.pop():
                    return False
                else:
                    slow = slow.next

            return True
```

Remove Linked List Elements

Remove all elements from a linked list of integers that have value val.

Example Given: 1 --> 2 --> 6 --> 3 --> 4 --> 5 --> 6, val = 6 Return: 1 --> 2 --> 3 --> 4 --> 5

URL: https://leetcode.com/problems/remove-linked-list-elements/

```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def removeElements(self, head, val):
        """
        :type head: ListNode
        :type val: int
        :rtype: ListNode
        """
        if head == None:
            return head
        elif head != None and head.next == None:
            if head.val == val:
                return None
            else:
                return head
        else:
            dummy = ListNode(0)
            dummy.next = head
            prev = dummy

            while head != None:
                if head.val == val:
                    prev.next = head.next
                    head = prev
                prev = head
                head = head.next

            return dummy.next
```

Remove Duplicates from Sorted Linked List

Given a sorted linked list, delete all duplicates such that each element appear only once.

For example, Given 1->1->2, return 1->2. Given 1->1->2->3->3, return 1->2->3.

URL: https://leetcode.com/problems/remove-duplicates-from-sorted-list/

```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None


class Solution(object):
    def deleteDuplicates(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        if head == None:
            return head
        elif head != None and head.next == None:
            return head
        else:
            lookup = {}
            current = head
            prev = head
            while current != None:
                if current.val in lookup:
                    prev.next = prev.next.next
                else:
                    lookup[current.val] = True
                    prev = current
                current = current.next

            return head
```

Remove Duplicates from Sorted Linked List II

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list.

For example, Given 1->2->3->3->4->4->5, return 1->2->5. Given 1->1->1->2->3, return 2->3.

URL: https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/

```python
# Definition for singly-linked list
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def deleteDuplicates(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        if head == None:
            return head
        else:
            dup_dict = {}
            current = head
            while current != None:
                if current.val in dup_dict:
                    dup_dict[current.val] += 1
                else:
                    dup_dict[current.val] = 1
                current = current.next

            list_values = []
            current = head
            while current != None:
```

```python
            if dup_dict[current.val] > 1:
                pass
            else:
                list_values.append(current.val)
            current = current.next
        if list_values == []:
            return None
        else:
            node1 = ListNode(list_values[0])
            head = node1
            for entries in list_values[1:]:
                new_node = ListNode(entries)
                node1.next = new_node
                node1 = new_node

            return head
```

Remove Nth node from End of List

Given a linked list, remove the nth node from the end of list and return its head.

For example,

Given linked list: 1->2->3->4->5, and n = 2.

After removing the second node from the end, the linked list becomes 1->2->3->5.
Note: Given n will always be valid. Try to do this in one pass.

URL: https://leetcode.com/problems/remove-nth-node-from-end-of-list/

```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def removeNthFromEnd(self, head, n):
        """
        :type head: ListNode
        :type n: int
        :rtype: ListNode
        """
        if head == None:
            return head
        else:
            dummy = ListNode(0)
            dummy.next = head
            fast = dummy
            slow = dummy
            for i in range(n):
                fast = fast.next

            while fast.next != None:
                fast = fast.next
                slow = slow.next

            slow.next = slow.next.next

            return dummy.next
```

Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment.

Design an algorithm to serialize and deserialize a binary tree. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary tree can be serialized to a string and this string can be deserialized to the original tree structure.

For example, you may serialize the following tree

```
1
  / \
 2   3
    / \
   4   5
```
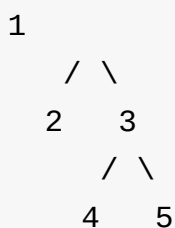
as

```
"[1,2,3,null,null,4,5]"
```

, just the same as

how LeetCode OJ serializes a binary tree

. You do not necessarily need to follow this format, so please be creative and come up with different approaches yourself.

Note: Do not use class member/global/static variables to store states. Your serialize and deserialize algorithms should be stateless.

URL: https://leetcode.com/problems/serialize-and-deserialize-binary-tree/

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

```python
class Codec:
    def __init__(self):
        self.serialized_array = []
        self.index = 0


    def serialize(self, root):
        """Encodes a tree to a single string.

        :type root: TreeNode
        :rtype: str
        """
        self.serialization_help(root)
        return self.serialized_array



    def serialization_help(self, root):
        if root == None:
            self.serialized_array.append(None)
            return
        self.serialized_array.append(root.val)
        self.serialize(root.left)
        self.serialize(root.right)

    def deserialize(self, data):
        """Decodes your encoded data to tree.

        :type data: str
        :rtype: TreeNode
        """
        if self.index == len(data) or data[self.index] == None:
            self.index += 1
            return None

        root = TreeNode(data[self.index])
        self.index += 1
        root.left = self.deserialize(data)
        root.right = self.deserialize(data)
        return root
```

```
# Your Codec object will be instantiated and called as such:
# codec = Codec()
# codec.deserialize(codec.serialize(root))
```

Preorder Traversal

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

For example: Given binary tree {1,#,2,3}, 1 \ 2 / 3 return [1,2,3].

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

URL: https://leetcode.com/problems/binary-tree-preorder-traversal/

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

class Solution:
    # @param {TreeNode} root
    # @return {integer[]}
    def preorderTraversal(self, root):
        if root == None:
            return []
        else:
            preorderList = []
            stack = []
            stack.append(root)
            while(stack != []):
                node = stack.pop()
                preorderList.append(node.val)
                if node.right:
                    stack.append(node.right)
                if node.left:
                    stack.append(node.left)
            return preorderList
```

BST Iterator

Implement an iterator over a binary search tree (BST). Your iterator will be initialized with the root node of a BST.

Calling next() will return the next smallest number in the BST.

Note: next() and hasNext() should run in average O(1) time and uses O(h) memory, where h is the height of the tree.

URL: https://leetcode.com/problems/binary-search-tree-iterator/

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

class BSTIterator:
    # @param root, a binary search tree's root node
    def __init__(self, root):
        self.stack = []
        node = root
        while node != None:
            self.stack.append(node)
            node = node.left


    # @return a boolean, whether we have a next smallest number
    def hasNext(self):
        return len(self.stack) != 0


    # @return an integer, the next smallest number
    def next(self):
        nextNode = self.stack.pop()
        currentNode = nextNode.right
        while currentNode != None:
            self.stack.append(currentNode)
            currentNode = currentNode.left
        return nextNode.val


# Your BSTIterator will be called like this:
# i, v = BSTIterator(root), []
# while i.hasNext(): v.append(i.next())
```

Inorder Traversal

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

For example: Given binary tree [1,null,2,3], 1 \ 2 / 3 return [1,3,2].

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

URL: https://leetcode.com/problems/binary-tree-inorder-traversal/

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

class Solution:
    # @param {TreeNode} root
    # @return {integer[]}
    def inorderTraversal(self, root):
        if root == None:
            return []
        else:
            result = []
            stack = []
            node = root
            while stack or node:
                if node:
                    stack.append(node)
                    node = node.left
                else:
                    node = stack.pop()
                    result.append(node.val)
                    node = node.right
            return result
```

Symmetric Tree

Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).

Note: Bonus points if you could solve it both recursively and iteratively.

URL: https://leetcode.com/problems/symmetric-tree/

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

class Solution(object):
    def isSymmetric(self, root):
        """
        :type root: TreeNode
        :rtype: bool
        """
        if root == None:
            return True
        else:
            return self.isMirror(root.left, root.right)


    def isMirror(self, root1, root2):
        if root1 == None and root2 == None:
            return True
        elif root1 == None or root2 == None:
            return False
        else:
            if root1.val == root2.val:
                return self.isMirror(root1.left, root2.right) an
d self.isMirror(root1.right, root2.left)
            else:
                return False
```

Balanced Binary Tree

Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of every node never differ by more than 1.

URL: https://leetcode.com/problems/balanced-binary-tree/

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

class Solution:
    # @param {TreeNode} root
    # @return {boolean}

    def getHeight(self, root):
        if root == None:
            return 0

        leftHeight = self.getHeight(root.left)
        if leftHeight == -1:
            return -1

        rightHeight = self.getHeight(root.right)
        if rightHeight == -1:
            return -1

        heightDiff = abs(leftHeight - rightHeight)
        if heightDiff > 1:
            return -1
        else:
            return max(leftHeight,rightHeight)+1


    def isBalanced(self, root):
        if self.getHeight(root) == -1:
            return False
        else:
            return True
```

Closest Binary Search Tree Value

Given a non-empty binary search tree and a target value, find the value in the BST that is closest to the target.

Note: Given target value is a floating point. You are guaranteed to have only one unique value in the BST that is closest to the target.

URL: https://leetcode.com/problems/closest-binary-search-tree-value/

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

class Solution(object):
    def closestValue(self, root, target):
        """
        :type root: TreeNode
        :type target: float
        :rtype: int
        """
        min_dif = float("inf")
        closestVal = None

        if root == None:
            return None
        else:
            while root:
                root_val = root.val
                val_dif = abs(root_val - target)
                if val_dif < min_dif:
                    min_dif = val_dif
                    closestVal = root_val
                if target < root_val:
                    if root.left != None:
                        root = root.left
                    else:
                        root = None
                else:
                    if root.right != None:
                        root = root.right
                    else:
                        root = None
        return closestVal
```

Postorder Traversal

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

For example: Given binary tree {1,#,2,3}, 1 \ 2 / 3 return [3,2,1].

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

URL: https://leetcode.com/problems/binary-tree-postorder-traversal/

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

class Solution(object):
    def postorderTraversal(self, root):
        """
        :type root: TreeNode
        :rtype: List[int]
        """
        if root == None:
            return []
        else:
            stack = []
            out_stack = []
            stack.append(root)

            while stack != []:
                current = stack.pop()
                out_stack.append(current.val)
                if current.left != None:
                    stack.append(current.left)
                if current.right != None:
                    stack.append(current.right)

            return out_stack[::-1]
```

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.

URL: https://leetcode.com/problems/maximum-depth-of-binary-tree/

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

class Solution(object):
    def maxDepth(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        if root == None:
            return 0
        else:
            return max(self.maxDepth(root.left), self.maxDepth(root.right)) + 1
```

Invert Binary Tree

Invert a binary tree.

```
        4
```

/ \ 2 7 / \ / \ 1 3 6 9 to 4 / \ 7 2 / \ / \ 9 6 3 1 Trivia: This problem was inspired by this original tweet by Max Howell: Google: 90% of our engineers use the software you wrote (Homebrew), but you can't invert a binary tree on a whiteboard so fuck off. URL: https://leetcode.com/problems/invert-binary-tree/

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

class Solution(object):
    def invertTree(self, root):
        """
        :type root: TreeNode
        :rtype: TreeNode
        """
        if root == None:
            return None
        else:
            stack = []
            stack.append(root)
            while stack != []:
                curr_node = stack.pop()
                if curr_node.left != None or curr_node.right !=
None:
                    temp = curr_node.left
                    curr_node.left = curr_node.right
                    curr_node.right = temp
                if curr_node.right != None:
                    stack.append(curr_node.right)
                if curr_node.left != None:
                    stack.append(curr_node.left)
            return root
```

Same Tree

Given two binary trees, write a function to check if they are equal or not.

Two binary trees are considered equal if they are structurally identical and the nodes have the same value.
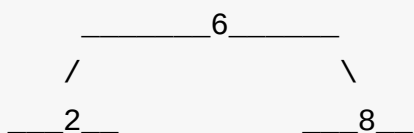
URL: https://leetcode.com/problems/same-tree/

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


class Solution:
    # @param {TreeNode} p
    # @param {TreeNode} q
    # @return {boolean}
    def isSameTree(self, p, q):
        if p == None and q == None:
            return True
        else:
            if p == None or q == None:
                return False
            else:
                if p.val == q.val:
                    return self.isSameTree(p.left, q.left) and self.isSameTree(p.right, q.right)
                else:
                    return False
```

Lowest Common Ancestor of a Binary Search Tree

Given a binary search tree (BST), find the lowest common ancestor (LCA) of two given nodes in the BST.

According to the definition of LCA on Wikipedia: "The lowest common ancestor is defined between two nodes v and w as the lowest node in T that has both v and w as descendants (where we allow a node to be a descendant of itself)."

```
      _____6_____
     /              \
  ___2__          ___8__
```

/ \ / \ 0 _4 7 9 / \ 3 5 For example, the lowest common ancestor (LCA) of nodes 2 and 8 is 6. Another example is LCA of nodes 2 and 4 is 2, since a node can be a descendant of itself according to the LCA definition.

URL: https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree/

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

class Solution(object):

    def __init__(self):
        self.inorder_list = []
        self.postorder_list = []

    def lowestCommonAncestor(self, root, p, q):
        """
```

```
        :type root: TreeNode
        :type p: TreeNode
        :type q: TreeNode
        :rtype: TreeNode
        """
        if root == None:
            return None
        else:
            self.inorder_traversal(root)
            self.postorder_traversal(root)
            #get the positions of node1 and node2 in the inorder
 traversal of the tree
            index_node1 = self.inorder_list.index(p.val)
            index_node2 = self.inorder_list.index(q.val)

            if index_node1 < index_node2:
                between_elems = self.inorder_list[index_node1 :
index_node2 + 1]
            else:
                between_elems = self.inorder_list[index_node2 :
index_node1 + 1]


            lca_elem = self.find_elem_max_index(between_elems)

            return lca_elem

    def find_elem_max_index(self, between_elems):
        max_index = -1
        elem = None
        for entries in between_elems:
            elem_index = self.postorder_list.index(entries)
            if elem_index > max_index:
                max_index = elem_index
                elem = entries
        return elem

    def inorder_traversal(self, node):
        if node:
            self.inorder_traversal(node.left)
```

```
            self.inorder_list.append(node.val)
            self.inorder_traversal(node.right)


    def postorder_traversal(self, node):
        if node:
            self.postorder_traversal(node.left)
            self.postorder_traversal(node.right)
            self.postorder_list.append(node.val)
```

Lowest Common Ancestor in a Binary Tree

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the definition of LCA on Wikipedia: "The lowest common ancestor is defined between two nodes v and w as the lowest node in T that has both v and w as descendants (where we allow a node to be a descendant of itself)."

```
        _____3_____
       /              \
    ___5__          ___1__
```
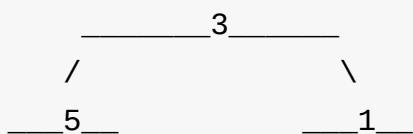
/ \ / \ 6 _2 0 8 / \ 7 4 For example, the lowest common ancestor (LCA) of nodes 5 and 1 is 3. Another example is LCA of nodes 5 and 4 is 5, since a node can be a descendant of itself according to the LCA definition.

URL: https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/

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

class Solution(object):

    def lowestCommonAncestor(self, root, p, q):
        """
        :type root: TreeNode
        :type p: TreeNode
        :type q: TreeNode
        :rtype: TreeNode
        """
        if root == None:
            return None

        if root == p or root == q:
            return root

        left = self.lowestCommonAncestor(root.left, p, q)
        right = self.lowestCommonAncestor(root.right, p, q)

        if left != None and right != None:
            return root

        if left == None:
            return right
        else:
            return left
```

Unique Binary Search Trees

Given n, how many structurally unique BST's (binary search trees) that store values 1...n?

For example, Given n = 3, there are a total of 5 unique BST's.

1 3 3 2 1 \ / / / \ \ 3 2 1 1 3 2 / / \ \ 2 1 2 3

URL: https://leetcode.com/problems/unique-binary-search-trees/

```python
class Solution(object):


    def numTrees(self, n):
        """
        :type n: int
        :rtype: int
        """
        solutions = [-1]*(n)
        return self.numUniqueBST(n, solutions)


    def numUniqueBST(self, n, solutions):

        if n < 0:
            return 0

        if n == 0 or n == 1:
            return 1

        possibilities = 0

        for i in range(0, n):
            if solutions[i] == -1:
                solutions[i] = self.numUniqueBST(i, solutions)
            if solutions[n-1-i] == -1:
                solutions[n-1-i] = self.numUniqueBST(n-1-i, solu
tions)
            possibilities += solutions[i]*solutions[n-1-i]
        return possibilities
```

Unique Binary Search Trees II

Given an integer n, generate all structurally unique BST's (binary search trees) that store values 1...n.

For example, Given n = 3, your program should return all 5 unique BST's shown below.

1 3 3 2 1 \ / / / \ \ 3 2 1 1 3 2 / / \ \ 2 1 2 3

URL:

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

class Solution(object):
    def generateTrees(self, n):
        """
        :type n: int
        :rtype: List[TreeNode]
        """
        if n == 0:
            return []
        else:
            return self.tree_constructor(1, n)


    def tree_constructor(self, m, n):
        results = []
        if m > n:
            results.append(None)
            return results

        for i in range(m, n+1):
            l = self.tree_constructor(m, i-1)
            r = self.tree_constructor(i+1, n)
            for left_trees in l:
                for right_trees in r:
                    curr_node = TreeNode(i)
                    curr_node.left = left_trees
                    curr_node.right = right_trees
                    results.append(curr_node)


        return results
```

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.

For 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.

URL: https://leetcode.com/problems/path-sum/

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

class Solution(object):

    def hasPathSum(self, root, sum):
        """
        :type root: TreeNode
        :type sum: int
        :rtype: bool
        """
        if root == None:
            return False
        else:
            current = root
            s = []
            s.append(current)
            s.append(current.val)

            while s != []:
                pathsum = s.pop()
                current = s.pop()

                if not current.left and not current.right:
```

```
                if pathsum == sum:
                    return True

            if current.right:
                rightpathsum = pathsum + current.right.val
                s.append(current.right)
                s.append(rightpathsum)

            if current.left:
                leftpathsum = pathsum + current.left.val
                s.append(current.left)
                s.append(leftpathsum)

        return False
```
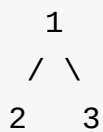
Binary Tree Maximum Path Sum

Given a binary tree, find the maximum path sum.

For this problem, a path is defined as any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The path does not need to go through the root.

For example: Given the below binary tree,

```
   1
  / \
 2   3
```

Return 6

URL: https://leetcode.com/problems/binary-tree-maximum-path-sum/

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

class Solution(object):
    def __init__(self):
        self.maxSum = -sys.maxint - 1


    # @param {TreeNode} root
    # @return {integer}
    def maxPathSum(self, root):
        self.findMax(root)
        return self.maxSum


    def findMax(self,root):
        if root == None:
            return 0
        left = self.findMax(root.left)
        right = self.findMax(root.right)
        self.maxSum = max(root.val + left + right, self.maxSum)
        ret = root.val + max(left,right)
        if ret < 0:
            return 0
        else:
            return ret
```

Binary Tree Level Order Traversal

Given a binary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

For example: Given binary tree [3,9,20,null,null,15,7], 3 / \ 9 20 / \ 15 7 return its level order traversal as: [[3], [9,20], [15,7]]

URL: https://leetcode.com/problems/binary-tree-level-order-traversal/

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

import Queue
class Solution:
    # @param {TreeNode} root
    # @return {integer[][]}
    def levelOrder(self, root):
        if root == None:
            return []
        else:
            q = Queue.Queue()
            q.put(root)
            q.put("#")
            levelOrderTraversal = []
            level = []
            while q.empty() == False:
                node = q.get()
                if node == "#":
                    if q.empty() == False:
                        q.put("#")
                    levelOrderTraversal.append(level)
                    level = []
                else:
                    level.append(node.val)
                    if node.left:
                        q.put(node.left)
                    if node.right:
                        q.put(node.right)

            return levelOrderTraversal
```

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 Binary tree [2,1,3], return true. Example 2: 1 / \ 2 3 Binary tree [1,2,3], return false.

URL: https://leetcode.com/problems/validate-binary-search-tree/

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
import sys
class Solution:
    def __init__(self):
        self.lastPrinted = -sys.maxsize-1
    # @param {TreeNode} root
    # @return {boolean}
    def isValidBST(self, root):
        if root == None:
            return True

        if self.isValidBST(root.left) == False:
            return False

        data = root.val
        if data <= self.lastPrinted:
            return False

        self.lastPrinted = data

        if self.isValidBST(root.right) == False:
            return False

        return True
```

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.

URL: https://leetcode.com/problems/minimum-depth-of-binary-tree/

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

class Solution(object):
    def minDepth(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        if root == None:
            return 0

        if root.left == None and root.right == None:
            return 1

        if root.left != None:
            left = self.minDepth(root.left)
        else:
            left = sys.maxsize

        if root.right != None:
            right = self.minDepth(root.right)
        else:
            right = sys.maxsize

        return 1 + min(left, right)
```

Convert Sorted Array to Binary Search Tree

Given an array where elements are sorted in ascending order, convert it to a height balanced BST.

URL: https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/

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

class Solution(object):
    def sortedArrayToBST(self, nums):
        """
        :type nums: List[int]
        :rtype: TreeNode
        """
        if nums == []:
            return None
        elif len(nums) == 1:
            return TreeNode(nums[0])
        else:
            start = 0
            end = len(nums) - 1
            return self.to_bst(nums, start, end)


    def to_bst(self, arr, start, end):
        if len(arr) == 0 or start > end:
            return None
        else:
            mid = (start + end) // 2
            node = TreeNode(arr[mid])
            node.left = self.to_bst(arr, start, mid - 1)
            node.right = self.to_bst(arr, mid + 1, end)
                return node`
```
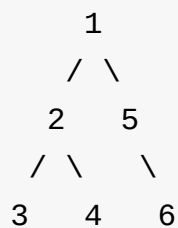
Flatten Binary Tree to Linked List

Given a binary tree, flatten it to a linked list in-place.

For example, Given

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

The flattened tree should look like: 1 \ 2 \ 3 \ 4 \ 5 \ 6

URL: https://leetcode.com/problems/flatten-binary-tree-to-linked-list/

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

class Solution(object):
    def flatten(self, root):
        """
        :type root: TreeNode
        :rtype: void Do not return anything, modify root in-plac
e instead.
        """
        if root == None:
            return root

        stack = []
        current = root

        while((stack != []) or (current != None)):

            if current.right != None:
                stack.append(current.right)

            if current.left != None:
                current.right = current.left
                current.left = None
            else:
                if stack != []:
                    temp = stack.pop()
                    current.right = temp

            current = current.right
```

Construct Binary Tree from Inorder and Preorder Traversal

Given preorder and inorder traversal of a tree, construct the binary tree.

Note: You may assume that duplicates do not exist in the tree.

URL: https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/

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

class Solution(object):
    def buildTree(self, preorder, inorder):
        """
        :type preorder: List[int]
        :type inorder: List[int]
        :rtype: TreeNode
        """
        if len(inorder) == 1:
            return TreeNode(inorder[0])
        return self.create_tree(inorder, 0, len(inorder) - 1, preorder, 0, len(preorder) - 1)

    def search_divindex(self, inorder, low_inorder, high_inorder, val):
        for i in range(low_inorder, high_inorder+1):
            if inorder[i] == val:
                return i
        return -1

    def create_tree(self, inorder, low_inorder, high_inorder, preorder, low_preorder, high_preorder):
```

```
        if (low_preorder > high_preorder) or (low_inorder > high
_inorder):
            return None

        root = TreeNode(preorder[low_preorder])
        div_index = self.search_divindex(inorder, low_inorder, h
igh_inorder, root.val)
        size_left_subtree = div_index - low_inorder
        size_right_subtree = high_inorder - div_index

        root.right = self.create_tree(inorder, div_index + 1, hi
gh_inorder, preorder,
                                      low_preorder + 1 + size_le
ft_subtree,
                                      low_preorder + size_left_s
ubtree + size_right_subtree)

        root.left = self.create_tree(inorder, low_inorder, div_i
ndex - 1, preorder,
                                      low_preorder + 1, low_preor
der + size_left_subtree)

        return root
```

Binary Tree Paths

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

For example, given the following binary tree:

1 / \ 2 3 \ 5 All root-to-leaf paths are:

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

URL: https://leetcode.com/problems/binary-tree-paths/

```python
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    # @param {TreeNode} root
    # @return {string[]}
    def binaryTreePaths(self, root):
        if root == None:
            return []
        else:
            paths = []
            current = root
            s = []
            s.append(current)
            s.append(str(current.val))


            while s != []:
                #pathsum = s.pop()
                path = s.pop()
                current = s.pop()

                if not current.left and not current.right:
                    paths.append(path)
```

```
            if current.right:
                rightstr = path + "->" + str(current.right.v
al)

                s.append(current.right)
                s.append(rightstr)


            if current.left:
                leftstr = path + "->" + str(current.left.val
)

                s.append(current.left)
                s.append(leftstr)
        return paths
```

Recover Binary Search Tree

Two elements of a binary search tree (BST) are swapped by mistake.

Recover the tree without changing its structure.

Note: A solution using O(n) space is pretty straight forward. Could you devise a constant space solution?

URL: https://leetcode.com/problems/recover-binary-search-tree/

```python
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None


class Solution(object):
    def __init__(self):
        self.__prev = None
        self.__node1 = None
        self.__node2 = None


    def recoverTree(self, root):
        """
        :type root: TreeNode
        :rtype: void Do not return anything, modify root in-plac
e instead.
        """
        self.recoverTreeHelp(root)
        temp = self.__node1.val
        self.__node1.val = self.__node2.val
        self.__node2.val = temp


    def recoverTreeHelp(self, root):
        if root == None:
            return None

        self.recoverTreeHelp(root.left)
        if self.__prev != None:
            if self.__prev.val > root.val:
                if self.__node1 == None:
                    self.__node1 = self.__prev
                self.__node2 = root
        self.__prev = root
        self.recoverTreeHelp(root.right)
```

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.

For 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]]

URL: https://leetcode.com/problems/path-sum-ii/

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

class Solution(object):
    def pathSum(self, root, sum):
        """
        :type root: TreeNode
        :type sum: int
        :rtype: List[List[int]]
        """
        if root == None:
            return []
        else:
            stack = []
            paths = []

            current = root
            stack.append(current)
            stack.append([current.val])
            stack.append(current.val)
            while stack != []:
                pathsum = stack.pop()
                path = stack.pop()
                curr = stack.pop()
```

```
            if curr.left == None and curr.right == None:
                if pathsum == sum:
                    paths.append(path)
            if curr.right:
                rightstr = path + [curr.right.val]
                rightsum = pathsum + curr.right.val
                stack.append(curr.right)
                stack.append(rightstr)
                stack.append(rightsum)
            if curr.left:
                leftstr = path + [curr.left.val]
                leftsum = pathsum + curr.left.val
                stack.append(curr.left)
                stack.append(leftstr)
                stack.append(leftsum)
        return paths
```

Binary Level Order Traversal II

Given a binary tree, return the bottom-up level order traversal of its nodes' values. (ie, from left to right, level by level from leaf to root).

For example: Given binary tree [3,9,20,null,null,15,7], 3 / \ 9 20 / \ 15 7 return its bottom-up level order traversal as: [[15,7], [9,20], [3]]

URL: https://leetcode.com/problems/binary-tree-level-order-traversal-ii/

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

import Queue
class Solution:
    # @param {TreeNode} root
    # @return {integer[][]}
    def levelOrderBottom(self, root):
        if root == None:
            return []
        else:
            q = Queue.Queue()
            q.put(root)
            q.put("#")
            levelOrderTraversal = []
            level = []
            stack = []

            while q.empty() == False:
                node = q.get()
                if node == "#":
                    if q.empty() == False:
                        q.put("#")
                    stack.append(level)
```

```
                level = []
            else:
                level.append(node.val)
                if node.left:
                    q.put(node.left)
                if node.right:
                    q.put(node.right)


        while stack:
            levelOrderTraversal.append(stack.pop())


        return levelOrderTraversal
```

Kth Smallest Element in a BST

Given a binary search tree, write a function kthSmallest to find the kth smallest element in it.

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

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?

Hint:

Try to utilize the property of a BST. What if you could modify the BST node's structure? The optimal runtime complexity is O(height of BST).

URL: https://leetcode.com/problems/kth-smallest-element-in-a-bst/

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

class Solution(object):
    def kthSmallest(self, root, k):
        """
        :type root: TreeNode
        :type k: int
        :rtype: int
        """
        if root == None:
            return None
        else:
            stack = []
            node = root
            count = 0
            while stack!= [] or node != None:
                if node != None:
                    stack.append(node)
                    node = node.left
                else:
                    inorder_node = stack.pop()
                    count += 1
                    if count == k:
                        return inorder_node.val
                    node = inorder_node.right
            return None
```

```python
class Solution(object):
    def kthSmallest(self, root, k):
        """
        :type root: TreeNode
        :type k: int
        :rtype: int
        """
        stack = []*k
        while True:
            while root:
                stack.append(root)
                root = root.left

            root = stack.pop()
            if k == 1:
                return root.val
            else:
                k -= 1
                root = root.right
```

Construct Binary Tree from Inorder and Postorder Traversal

Given inorder and postorder traversal of a tree, construct the binary tree.

Note: You may assume that duplicates do not exist in the tree.

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


class Solution(object):
    def buildTree(self, inorder, postorder):
        """
        :type inorder: List[int]
        :type postorder: List[int]
        :rtype: TreeNode
        """
        return self.create_tree(inorder, 0, len(inorder) -1 , postorder, 0, len(postorder) - 1)


    def search_divindex(self, inorder, low_inorder, high_inorder, val):
        for i in range(low_inorder, high_inorder+1):
            if inorder[i] == val:
                return i
        return -1

    def create_tree(self, inorder, low_inorder, high_inorder, postorder, low_postorder, high_postorder):
        if (low_inorder > high_inorder) or (low_postorder > high_postorder):
            return None
```

```python
        root = TreeNode(postorder[high_postorder])

        div_index = self.search_divindex(inorder, low_inorder, h
igh_inorder, root.val)

        size_left_subtree = div_index - low_inorder
        size_right_subtree = high_inorder - div_index

        root.right = self.create_tree(inorder, div_index + 1, hi
gh_inorder, postorder,
                                      high_postorder - size_righ
t_subtree, high_postorder - 1)

        root.left = self.create_tree(inorder, low_inorder, div_i
ndex - 1,
                                     postorder,
                                     high_postorder - size_right
_subtree - size_left_subtree,
                                     high_postorder - size_right
_subtree - 1)

        return root
```

Binary Tree Right Side View

Given a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom.

For example: Given the following binary tree, 1 <--- / \ 2 3 <--- \ \ 5 4 <--- You should return [1, 3, 4].

URL: https://leetcode.com/problems/binary-tree-right-side-view/

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

import Queue
class Solution:
    # @param {TreeNode} root
    # @return {integer[]}
    def rightSideView(self, root):
        if root == None:
            return []
        else:
            q = Queue.Queue()
            q.put(root)
            q.put("#")
            rightSideView = []
            level = []
            while q.empty() == False:
                node = q.get()
                if node == "#":
                    if q.empty() == False:
                        q.put("#")
                    rightSideView.append(level[-1])
                    level = []
                else:
                    level.append(node.val)
                    if node.left != None:
                        q.put(node.left)
                    if node.right != None:
                        q.put(node.right)
            return rightSideView
```

Sum Root to Leaf Numbers

Given a binary tree containing digits from 0-9 only, each root-to-leaf path could represent a number.

An example is the root-to-leaf path 1->2->3 which represents the number 123.

Find the total sum of all root-to-leaf numbers.

For example,

```
   1
```

/ \ 2 3 The root-to-leaf path 1->2 represents the number 12. The root-to-leaf path 1->3 represents the number 13.

Return the sum = 12 + 13 = 25.

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

class Solution(object):
    def sumNumbers(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        if root == None:
            return 0
        else:
            stack = []
            paths = []
            stack.append(root)
            stack.append(str(root.val))
            while stack != []:
                path = stack.pop()
                current = stack.pop()
                if current.left == None and current.right == None:
                    paths.append(int(path))
                if current.right:
                    rightstr = path + str(current.right.val)
                    stack.append(current.right)
                    stack.append(rightstr)
                if current.left:
                    leftstr = path + str(current.left.val)
                    stack.append(current.left)
                    stack.append(leftstr)
            return sum(paths)
```

Binary Tree Zigzag Level Order Traversal

Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).

For example: Given binary tree [3,9,20,null,null,15,7], 3 / \ 9 20 / \ 15 7 return its zigzag level order traversal as: [[3], [20,9], [15,7]]

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

import Queue
class Solution:
    # @param {TreeNode} root
    # @return {integer[][]}
    def zigzagLevelOrder(self, root):
        if root == None:
            return []
        else:
            q = Queue.Queue()
            q.put(root)
            q.put("#")
            levelOrderTraversal = []
            level = []
            levelNo = 0
            while q.empty() == False:
                node = q.get()

                if node == "#":
                    if q.empty() == False:
                        q.put("#")

                        if levelNo == 0 or levelNo % 2 == 0:
                            levelOrderTraversal.append(level)
```

```python
                else:
                    levelOrderTraversal.append(level[::-1])
                level = []
                levelNo += 1
            else:
                level.append(node.val)
                if node.left:
                    q.put(node.left)
                if node.right:
                    q.put(node.right)

    return levelOrderTraversal
```

House Robber III

The thief has found himself a new place for his thievery again. There is only one entrance to this area, called the "root." Besides the root, each house has one and only one parent house. After a tour, the smart thief realized that "all houses in this place forms a binary tree". It will automatically contact the police if two directly-linked houses were broken into on the same night.

Determine the maximum amount of money the thief can rob tonight without alerting the police.

Example 1: 3 / \ 2 3 \ \ 3 1 Maximum amount of money the thief can rob = 3 + 3 + 1 = 7. Example 2: 3 / \ 4 5 / \ \ 1 3 1 Maximum amount of money the thief can rob = 4 + 5 = 9.

URL: https://leetcode.com/problems/house-robber-iii/

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

class Solution(object):
    def rob(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        if root == None:
            return 0
        else:
            result = self.rob_max(root)
            return max(result[0], result[1])


    def rob_max(self, root):
        if root == None:
            return [0, 0]
        else:
            left_res = self.rob_max(root.left)
            right_res = self.rob_max(root.right)
            result = [0]*2
            result[0] = root.val + left_res[1] + right_res[1]
            result[1] = max(left_res[0], left_res[1]) + max(right_res[0], right_res[1])
            return result
```

Inorder Successor in BST

Given a binary search tree and a node in it, find the in-order successor of that node in the BST.

Note: If the given node has no in-order successor in the tree, return null.

URL: https://leetcode.com/problems/inorder-successor-in-bst/

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

class Solution(object):
    def inorderSuccessor(self, root, p):
        """
        :type root: TreeNode
        :type p: TreeNode
        :rtype: TreeNode
        """
        successor = None

        while root != None and root.val != p.val:
            if root.val > p.val:
                successor = root
                root = root.left
            else:
                root = root.right

        if root == None:
            return None

        if root.right == None:
            return successor

        root = root.right
        while root.left != None:
            root = root.left
        return root
```

Binary Tree Longest Consecutive Sequence

Given a binary tree, find the length of the longest consecutive sequence path.

The path refers to any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The longest consecutive path need to be from parent to child (cannot be the reverse).

For example, 1 \ 3 / \ 2 4 \ 5 Longest consecutive sequence path is 3-4-5, so return 3. 2 \ 3 / 2
/ 1 Longest consecutive sequence path is 2-3,not3-2-1, so return 2.

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

from Queue import Queue
import sys
class Solution(object):
    def longestConsecutive(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        if root == None:
            return 0
        if root.right == None and root.left == None:
            return 1
        else:
            max_size = 1
            size_q = Queue()
            node_q = Queue()
            node_q.put(root)
```

```
            size_q.put(1)
        while node_q.empty() == False:
            curr_node = node_q.get()
            curr_size = size_q.get()

            if curr_node.left:
                left_size = curr_size
                if curr_node.val == curr_node.left.val - 1:
                    left_size += 1
                    max_size = max(max_size, left_size)
                else:
                    left_size = 1

                node_q.put(curr_node.left)
                size_q.put(left_size)

            if curr_node.right:
                right_size = curr_size
                if curr_node.val == curr_node.right.val - 1:
                    right_size += 1
                    max_size = max(max_size, right_size)
                else:
                    right_size = 1

                node_q.put(curr_node.right)
                size_q.put(right_size)

        return max_size
```

Verify Preorder Sequence in Binary Search Tree

Given an array of numbers, verify whether it is the correct preorder traversal sequence of a binary search tree.

You may assume each number in the sequence is unique.

Follow up: Could you do it using only constant space complexity?

URL: https://leetcode.com/problems/verify-preorder-sequence-in-binary-search-tree/

```python
import sys
class Solution(object):
    def verifyPreorder(self, preorder):
        """
        :type preorder: List[int]
        :rtype: bool
        """
        stack = []
        root = -sys.maxsize-1

        for entries in preorder:

            if entries < root:
                return False

            while stack != [] and stack[-1] < entries:
                root = stack.pop()

            stack.append(entries)

        return True
```

Binary Tree Upside Down

Given a binary tree where all the right nodes are either leaf nodes with a sibling (a left node that shares the same parent node) or empty, flip it upside down and turn it into a tree where the original right nodes turned into left leaf nodes. Return the new root.

For example: Given a binary tree {1,2,3,4,5}, 1 / \ 2 3 / \ 4 5 return the root of the binary tree [4,5,2,#,#,3,1]. 4 / \ 5 2 / \ 3 1

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

class Solution(object):
    def upsideDownBinaryTree(self, root):
        """
        :type root: TreeNode
        :rtype: TreeNode
        """
        p = root
        parent = None
        parent_right = None

        while p:
            left = p.left
            p.left = parent_right
            parent_right = p.right
            p.right = parent
            parent = p
            p = left

        return parent
```

Count Univalue Subtrees

Given a binary tree, count the number of uni-value subtrees.

A Uni-value subtree means all nodes of the subtree have the same value.

For example: Given binary tree, 5 / \ 1 5 / \ \ 5 5 5 return 4.

URL: https://leetcode.com/problems/count-univalue-subtrees/

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

class Solution(object):
    def __init__(self):
        self.__count = 0


    def countUnivalSubtrees(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        self.count_univalue_subtrees(root)
        return self.__count


    def count_univalue_subtrees(self, root):
        if root == None:
            return True
        if root.left == None and root.right == None:
            self.__count += 1
            return True
        left = self.count_univalue_subtrees(root.left)
        right = self.count_univalue_subtrees(root.right)

        if (left and right) and (root.left == None or root.left.
val == root.val) and (root.right == None or root.right.val == ro
ot.val):
            self.__count += 1
            return True
        else:
            return False
```

Number of Connected Components in an Undirected Graph

Given n nodes labeled from 0 to n - 1 and a list of undirected edges (each edge is a pair of nodes), write a function to find the number of connected components in an undirected graph.

URL : https://leetcode.com/problems/number-of-connected-components-in-an-undirected-graph/

```python
import sys
from queue import Queue


class Vertex:
    def __init__(self, node):
        self.id = node
        self.adjacent = {}
        # Set distance to infinity for all nodes
        self.distance = sys.maxsize
        # Mark all nodes unvisited
        self.visited = False
        # Mark all nodes color with white
        self.color = 'white'
        # Predecessor
        self.previous = None

    def addNeighbor(self, neighbor, weight=0):
        self.adjacent[neighbor] = weight

    def getConnections(self):
        return self.adjacent.keys()

    def getVertexID(self):
        return self.id

    def getWeight(self, neighbor):
        return self.adjacent[neighbor]
```

```python
    def setDistance(self, dist):
        self.distance = dist

    def getDistance(self):
        return self.distance

    def setColor(self, color):
        self.color = color

    def getColor(self):
        return self.color

    def setPrevious(self, prev):
        self.previous = prev

    def setVisited(self):
        self.visited = True

    def __str__(self):
        return str(self.id) + ' adjacent: ' + str([x.id for x in
 self.adjacent])

class Graph:
    def __init__(self):
        self.vertDictionary = {}
        self.numVertices = 0

    def __iter__(self):
        return iter(self.vertDictionary.values())

    def addVertex(self, node):
        self.numVertices = self.numVertices + 1
        newVertex = Vertex(node)
        self.vertDictionary[node] = newVertex
        return newVertex

    def getVertex(self, n):
        if n in self.vertDictionary:
            return self.vertDictionary[n]
```

```python
        else:
            return None


    def addEdge(self, frm, to, cost=0):
        if frm not in self.vertDictionary:
            self.addVertex(frm)
        if to not in self.vertDictionary:
            self.addVertex(to)

        self.vertDictionary[frm].addNeighbor(self.vertDictionary
[to], cost)
        self.vertDictionary[to].addNeighbor(self.vertDictionary[
frm], cost)


    def getVertices(self):
        return self.vertDictionary.keys()


    def setPrevious(self, current):
        self.previous = current


    def getPrevious(self, current):
        return self.previous


class Solution(object):
    def countComponents(self, n, edges):
        """
        :type n: int
        :type edges: List[List[int]]
        :rtype: int
        """
        if n == 1 and edges == []:
            return 1
        else:
            G = Graph()
            for entries in edges:
                G.addEdge(entries[0], entries[1], 1)
            count = 0
            for vertex in G:
                if vertex.getColor() == "white":
                    count += 1
```

```
                    self.bfs(vertex)

            return count

    def bfs(self, vertex):
        vertex.setColor("gray")
        q = Queue()
        q.put(vertex)
        while q.empty() == False:
            curr_node = q.get()
            for nbr in curr_node.getConnections():
                if nbr.getColor() == "white":
                    nbr.setColor("gray")
                    q.put(nbr)
            curr_node.setColor("black")


if __name__ == "__main__":

    n = 5
    edges1 = [[0, 1], [1, 2], [3, 4]]
    edges2 = [[0, 1], [1, 2], [2, 3], [3, 4]]

    soln = Solution()
    print(soln.countComponents(n, edges1))
    print(soln.countComponents(n, edges2))
```

Course Schedule

There are a total of n courses you have to take, labeled from 0 to n - 1.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: [0,1]

Given the total number of courses and a list of prerequisite pairs, is it possible for you to finish all courses?

For example:

2, [[1,0]] There are a total of 2 courses to take. To take course 1 you should have finished course 0. So it is possible.

2, [[1,0],[0,1]] There are a total of 2 courses to take. To take course 1 you should have finished course 0, and to take course 0 you should also have finished course 1. So it is impossible.

Note: The input prerequisites is a graph represented by a list of edges, not adjacency matrices. Read more about how a graph is represented.

URL: https://leetcode.com/problems/course-schedule/

```python
class Vertex:

    def __init__(self, key):
        self.id = key
        self.adjacent = {}
        self.indegree = 0
        self.outdegree = 0
        self.predecessor = None
        self.visit_time = 0
        self.finish_time = 0
        self.color = "white"


    def add_neighbor(self, nbr, weight=0):
        self.adjacent[nbr] = weight
```

```python
    def get_neighbors(self):
        return self.adjacent.keys()

    def get_id(self):
        return self.id

    def get_weight(self, nbr):
        return self.adjacent[nbr]

    def get_indegree(self):
        return self.indegree

    def set_indegree(self, indegree):
        self.indegree = indegree

    def get_outdegree(self):
        return self.outdegree

    def set_outdegree(self, outdegree):
        self.outdegree = outdegree

    def get_predecessor(self):
        return self.predecessor

    def set_predecessor(self, pred):
        self.predecessor = pred

    def get_visit_time(self):
        return self.visit_time

    def set_visit_time(self, visit_time):
        self.visit_time = visit_time

    def get_finish_time(self):
        return self.finish_time

    def set_finish_time(self, finish_time):
        self.finish_time = finish_time

    def get_color(self):
```

```python
            return self.color

    def set_color(self, color):
        self.color = color

    def __str__(self):
        return str(self.id) + ' connectedTo: ' + str([x.id for x
 in self.adjacent])




class Graph:

    def __init__(self):
        self.vertex_dict = {}
        self.no_vertices = 0
        self.no_edges = 0

    def add_vertex(self, vert_key):
        new_vertex_obj = Vertex(vert_key)
        self.vertex_dict[vert_key] = new_vertex_obj
        self.no_vertices += 1

    def get_vertex(self, vert_key):
        if vert_key in self.vertex_dict:
            return self.vertex_dict[vert_key]
        else:
            return None

    def add_edge(self, fro, to, weight=1):
        if fro not in self.vertex_dict:
            self.add_vertex(fro)
            from_vertex = self.get_vertex(fro)
        else:
            from_vertex = self.vertex_dict[fro]

        if to not in self.vertex_dict:
            self.add_vertex(to)
            to_vertex = self.get_vertex(to)
```

```python
        else:
            to_vertex = self.vertex_dict[to]

        from_vertex.add_neighbor(to_vertex, weight)
        from_vertex.set_outdegree(from_vertex.get_outdegree() +
1)
        to_vertex.set_indegree(to_vertex.get_indegree() + 1)
        self.no_edges += 1

    def get_edges(self):
        edges = []
        for u in self.vertex_dict:
            for v in self.vertex_dict[u].get_neighbors():
                u_id = u
                #print(v)
                v_id = v.get_id()
                edges.append((u_id, v_id, self.vertex_dict[u].ge
t_weight(v)))
        return edges

    def get_vertices(self):
        return self.vertex_dict


class DFS:

    def __init__(self, graph):
        self.graph = graph
        self.has_cycle = False

    def dfs(self):
        for vertex in self.graph.get_vertices():
            if self.graph.vertex_dict[vertex].get_color() == "wh
ite":
                self.dfs_visit(self.graph.vertex_dict[vertex])

    def dfs_visit(self, node):
        node.set_color("gray")
        for vert in node.get_neighbors():
            if vert.get_color() == "gray":
```

```python
                self.has_cycle = True
            if vert.get_color() == "white":
                vert.set_color("gray")
                self.dfs_visit(vert)
        node.set_color("black")


class Solution(object):
    def canFinish(self, numCourses, prerequisites):
        """
        :type numCourses: int
        :type prerequisites: List[List[int]]
        :rtype: bool
        """
        if not prerequisites:
            return True
        else:
            g = Graph()

            for edge in prerequisites:
                g.add_edge(edge[0], edge[1])

            dfs_obj = DFS(g)
            dfs_obj.dfs()
            if dfs_obj.has_cycle == True:
                return False
            else:
                return True


if __name__ == "__main__":
    soln1 = Solution()
    print(soln1.canFinish(2, [[1,0]]))


    soln2 = Solution()
    print(soln2.canFinish(2, [[1,0],[0,1]]))
```

Graph Valid Tree

Given n nodes labeled from 0 to n - 1 and a list of undirected edges (each edge is a pair of nodes), write a function to check whether these edges make up a valid tree.

For example:

Given n = 5 and edges = [[0, 1], [0, 2], [0, 3], [1, 4]], return true.

Given n = 5 and edges = [[0, 1], [1, 2], [2, 3], [1, 3], [1, 4]], return false.

Hint:

Given n = 5 and edges = [[0, 1], [1, 2], [3, 4]], what should your return? Is this case a valid tree? According to the definition of tree on Wikipedia: "a tree is an undirected graph in which any two vertices are connected by exactly one path. In other words, any connected graph without simple cycles is a tree."

URL: https://leetcode.com/problems/graph-valid-tree/

```python
import sys
class Vertex:
    def __init__(self, node):
        self.id = node
        self.adjacent = {}
        # Set distance to infinity for all nodes
        self.distance = sys.maxsize
        # Mark all nodes unvisited
        self.visited = False
        # Mark all nodes color with white
        self.color = 'white'
        # Predecessor
        self.previous = None

    def addNeighbor(self, neighbor, weight=0):
        self.adjacent[neighbor] = weight


    def getConnections(self):
```

```
            return self.adjacent.keys()

    def getVertexID(self):
        return self.id

    def getWeight(self, neighbor):
        return self.adjacent[neighbor]

    def setDistance(self, dist):
        self.distance = dist

    def getDistance(self):
        return self.distance

    def setColor(self, color):
        self.color = color

    def getColor(self):
        return self.color

    def setPrevious(self, prev):
        self.previous = prev

    def setVisited(self):
        self.visited = True

    def __str__(self):
        return str(self.id) + ' adjacent: ' + str([x.id for x in
 self.adjacent])

class Graph:
    def __init__(self):
        self.vertDictionary = {}
        self.numVertices = 0

    def __iter__(self):
        return iter(self.vertDictionary.values())

    def addVertex(self, node):
        self.numVertices = self.numVertices + 1
```

```python
            newVertex = Vertex(node)
            self.vertDictionary[node] = newVertex
            return newVertex

    def getVertex(self, n):
        if n in self.vertDictionary:
            return self.vertDictionary[n]
        else:
            return None

    def addEdge(self, frm, to, cost=0):
        if frm not in self.vertDictionary:
            self.addVertex(frm)
        if to not in self.vertDictionary:
            self.addVertex(to)

        self.vertDictionary[frm].addNeighbor(self.vertDictionary[to], cost)
        self.vertDictionary[to].addNeighbor(self.vertDictionary[frm], cost)

    def getVertices(self):
        return self.vertDictionary.keys()

    def setPrevious(self, current):
        self.previous = current

    def getPrevious(self, current):
        return self.previous

class Solution:
    def validTree(self, n, edges):
        """
        :type n: int
        :type edges: List[List[int]]
        :rtype: bool
        """
        if n == 1 and len(edges) == 0:
            return True
        elif self.check_input(n, edges) == False:
```

```python
                return False
        elif n == 0 and len(edges) > 0:
            return False
        elif n == 1 and len(edges) >= 1:
            return False
        else:
            G = Graph()
            for entries in edges:
                G.addEdge(entries[0], entries[1], 1)

            results = []
            for vertex in G:
                if vertex.getColor() == "white":
                    results.append(self.check_validity(vertex))

            if len(results) > 1:
                return False
            else:
                return results[0]


    def check_input(self, n, edges):
        vertices = []
        for entries in edges:
            vertices.append(entries[0])
            vertices.append(entries[1])
        if len(set(vertices)) != n:
            return False
        else:
            return True

    def check_validity(self, start):
        stack = []
        start.setColor("gray")
        stack.append(start)
        while stack != []:
            curr_node = stack.pop()
            for nbr in curr_node.getConnections():
                if nbr.getColor() == "gray":
                    return False
```

```
            if nbr.getColor() == "white":
                nbr.setColor("gray")
                stack.append(nbr)
        curr_node.setColor("black")
    return True
```

Course Schedule 2

There are a total of n courses you have to take, labeled from 0 to n - 1.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: [0,1]

Given the total number of courses and a list of prerequisite pairs, return the ordering of courses you should take to finish all courses.

There may be multiple correct orders, you just need to return one of them. If it is impossible to finish all courses, return an empty array.

For example:

2, [[1,0]] There are a total of 2 courses to take. To take course 1 you should have finished course 0. So the correct course order is [0,1]

4, [[1,0],[2,0],[3,1],[3,2]] There are a total of 4 courses to take. To take course 3 you should have finished both courses 1 and 2. Both courses 1 and 2 should be taken after you finished course 0. So one correct course order is [0,1,2,3]. Another correct ordering is [0,2,1,3].

URL: https://leetcode.com/problems/course-schedule-ii/

```python
from queue import Queue
import sys


class Vertex:
    def __init__(self, node):
        self.id = node
        self.adjacent = {}
        # Set distance to infinity for all nodes
        self.distance = sys.maxsize
        # Mark all nodes unvisited
        self.visited = False
        # Mark all nodes color with white
        self.color = 'white'
        # Predecessor
```

```python
        self.previous = None
        #indegree of the vertex
        self.indegree = 0

    def addNeighbor(self, neighbor, weight=0):
        self.adjacent[neighbor] = weight

    def getConnections(self):
        return self.adjacent.keys()

    def getVertexID(self):
        return self.id

    def getWeight(self, neighbor):
        return self.adjacent[neighbor]

    def setDistance(self, dist):
        self.distance = dist

    def getDistance(self):
        return self.distance

    def setColor(self, color):
        self.color = color

    def getColor(self):
        return self.color

    def setPrevious(self, prev):
        self.previous = prev

    def setVisited(self):
        self.visited = True

    def setIndegree(self, indegree):
        self.indegree = indegree

    def getIndegree(self):
        return self.indegree
```

```python
    def __str__(self):
        return str(self.id) + ' adjacent: ' + str([x.id for x in
  self.adjacent])


class DirectedGraph:
    def __init__(self):
        self.vertDictionary = {}
        self.numVertices = 0


    def __iter__(self):
        return iter(self.vertDictionary.values())


    def addVertex(self, node):
        self.numVertices = self.numVertices + 1
        newVertex = Vertex(node)
        self.vertDictionary[node] = newVertex
        return newVertex


    def getVertex(self, n):
        if n in self.vertDictionary:
            return self.vertDictionary[n]
        else:
            return None


    def addEdge(self, frm, to, cost=0):
        if frm not in self.vertDictionary:
            self.addVertex(frm)
        if to not in self.vertDictionary:
            self.addVertex(to)

        self.vertDictionary[frm].addNeighbor(self.vertDictionary
[to], cost)
        self.vertDictionary[to].setIndegree(self.vertDictionary[
to].getIndegree() + 1)


    def getVertices(self):
        return self.vertDictionary.keys()


    def setPrevious(self, current):
        self.previous = current
```

```python
    def getPrevious(self, current):
        return self.previous



class Solution:
    def __init__(self):
        self.has_cycle = False


    def findOrder(self, numCourses, prerequisites):
        """
        :type numCourses: int
        :type prerequisites: List[List[int]]
        :rtype: List[int]
        """
        if prerequisites == [] and numCourses > 0:
            return [entries for entries in range(numCourses)]
        elif prerequisites == [] and numCourses == 0:
            return []
        else:
            G = DirectedGraph()
            for entries in prerequisites:
                G.addEdge(entries[1], entries[0], 1)

            return self.topsort(G)


    def topsort(self, G):
        if G.getVertices() == []:
            return []
        else:
            topological_list = []
            topological_queue = Queue()
            nodes = G.getVertices()
            for node in G:
                if node.getIndegree() == 0:
                    topological_queue.put(node)

            while topological_queue.empty() == False:
                curr_node = topological_queue.get()
                topological_list.append(curr_node.getVertexID())
```

```python
                for nbr in curr_node.getConnections():
                    nbr.setIndegree(nbr.getIndegree() - 1)
                    if nbr.getIndegree() == 0:
                        topological_queue.put(nbr)

            if len(topological_list) != len(nodes):
                self.has_cycle = True

            return topological_list


if __name__ == "__main__":

    soln = Solution()
    print(soln.findOrder(4, [[1,0],[2,0],[3,1],[3,2]]))
    print(soln.findOrder(2, [[1,0]]))
    print(soln.findOrder(3, [[1,0]]))
```

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:

11110 11010 11000 00000 Answer: 1

Example 2:

11000 11000 00100 00011 Answer: 3

URL: https://leetcode.com/problems/number-of-islands/

```python
class Solution:
    # @param {boolean[][]} grid a boolean 2D matrix
    # @return {int} an integer
    def numIslands(self, grid):
        if not grid:
            return 0

        row = len(grid)
        col = len(grid[0])
        used = [[False for j in xrange(col)] for i in xrange(row
)]

        count = 0
        for i in xrange(row):
            for j in xrange(col):
                if grid[i][j] == '1' and not used[i][j]:
                    self.dfs(grid, used, row, col, i, j)
                    count += 1
        return count

    def dfs(self, grid, used, row, col, x, y):
        if grid[x][y] == '0' or used[x][y]:
            return
        used[x][y] = True

        if x != 0:
            self.dfs(grid, used, row, col, x - 1, y)
        if x != row - 1:
            self.dfs(grid, used, row, col, x + 1, y)
        if y != 0:
            self.dfs(grid, used, row, col, x, y - 1)
        if y != col - 1:
            self.dfs(grid, used, row, col, x, y + 1)
```

Merge K Sorted Linked Lists

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

URL: https://leetcode.com/problems/merge-k-sorted-lists/

```python
import heapq
# Definition for singly-linked list.
class ListNode(object):
    def __init__(self, x):
        self.val = x
        self.next = None


class Solution(object):
    def mergeKLists(self, lists):
        """
        :type lists: List[ListNode]
        :rtype: ListNode
        """
        if lists == [] or lists == None:
            return None
        else:
            pq = []
            for i in range(len(lists)):
                if lists[i] != None:
                    item = (lists[i].val, i, lists[i])
                    heapq.heappush(pq, item)

            dummy = ListNode(0)
            p = dummy

            while pq != []:
                heap_item = heapq.heappop(pq)
                p.next = heap_item[2]
                p = p.next
                if heap_item[2].next != None:
                    item = (heap_item[2].next.val, heap_item[1],
 heap_item[2].next)
                    heapq.heappush(pq, item)

            return dummy.next
```

Kth Largest Element in an Array

Find the kth largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

For example, Given [3,2,1,5,6,4] and k = 2, return 5.

Note: You may assume k is always valid, 1 ≤ k ≤ array's length

URL: https://leetcode.com/problems/kth-largest-element-in-an-array/

```python
class Solution(object):
    def findKthLargest(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: int
        """
        if nums == []:
            return nums
        else:
            heap = []
            for i in range(0, k):
                heapq.heappush(heap, (nums[i], i))

            for j in range(k, len(nums)):
                root_element = [entries for entries in heapq.nsm
allest(1, heap)][0]
                index_root_element = heap.index(root_element)
                if nums[j] > root_element[0]:
                    heap[index_root_element] = (nums[j], j)
                    heapq.heapify(heap)

            return heapq.heappop(heap)[0]
```

2 Sum II

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. Please note that your returned answers (both index1 and index2) are not zero-based.

You may assume that each input would have exactly one solution.

Input: numbers={2, 7, 11, 15}, target=9 Output: index1=1, index2=2

URL: https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/

```python
class Solution(object):
    def twoSum(self, numbers, target):
        """
        :type numbers: List[int]
        :type target: int
        :rtype: List[int]
        """
        if len(numbers) == 0:
            return [-1]
        else:
            start = 0
            end = len(numbers) - 1
            while start < end:
                curr_sum = numbers[start] + numbers[end]
                if curr_sum == target:
                    return [start+1, end+1]
                elif curr_sum < target:
                    start += 1
                elif curr_sum > target:
                    end -= 1
            return [-1]
```

2 Sum III

Design and implement a TwoSum class. It should support the following operations: add and find.

add - Add the number to an internal data structure. find - Find if there exists any pair of numbers which sum is equal to the value.

For example, add(1); add(3); add(5); find(4) -> true find(7) -> false

URL: https://leetcode.com/problems/two-sum-iii-data-structure-design/

```python
import collections


class TwoSum(object):

    def __init__(self):
        """
        initialize your data structure here
        """
        self.__num_list = collections.defaultdict(int)



    def add(self, number):
        """
        Add the number to an internal data structure.
        :rtype: nothing
        """
        self.__num_list[number] += 1



    def find(self, value):
        """
        Find if there exists any pair of numbers which sum is eq
 ual to the value.
        :type value: int
        :rtype: bool
        """
```

```
        if len(self.__num_list) == 0:
            return False
        else:
            for entries in self.__num_list.keys():
                target = value - entries
                if (target in self.__num_list) and (entries != t
arget or self.__num_list[target] > 1):
                    return True
            return False


if __name__ == "__main__":
    # Your TwoSum object will be instantiated and called as such
:
    twoSum = TwoSum()
    twoSum.add(0)
    twoSum.add(0)
    print(twoSum.find(0))
```

Contains Duplicate

Given an array of integers, find if the array contains any duplicates. Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

URL: https://leetcode.com/problems/contains-duplicate/

```python
class Solution(object):
    def containsDuplicate(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """
        if not nums:
            return False
        elif len(nums) == 1:
            return False
        else:
            dup_dict = {}

            for entries in nums:
                if entries in dup_dict:
                    return True
                else:
                    dup_dict[entries] = 1
            return False
```

Rotate Array

Rotate an array of n elements to the right by k steps.

For example, with n = 7 and k = 3, the array [1,2,3,4,5,6,7] is rotated to [5,6,7,1,2,3,4].

Note: Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem.

URL: https://leetcode.com/problems/rotate-array/

```python
class Solution(object):
    def rotate(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: void Do not return anything, modify nums in-plac
e instead.
        """
        n = len(nums)
        if n < 2 or k == 0:
            pass
        else:
            if k >= n:
                k = k % n
            a = n - k
            self.reverse(nums, 0, a-1)
            self.reverse(nums, a, n-1)
            self.reverse(nums, 0, n-1)


    def reverse(self, nums, start, end):
        i = start
        j = end
        while i < j:
            nums[i], nums[j] = nums[j], nums[i]
            i += 1
            j -= 1

if __name__ == "__main__":
    soln = Solution()
    nums = [1,2,3,4,5,6,7]
    soln.rotate(nums, 3)
    print(nums)
```

3 Sum Smaller

Given an array of n integers nums and a target, find the number of index triplets i, j, k with 0 <= i < j < k < n that satisfy the condition nums[i] + nums[j] + nums[k] < target.

For example, given nums = [-2, 0, 1, 3], and target = 2.

Return 2. Because there are two triplets which sums are less than 2:

[-2, 0, 1] [-2, 0, 3]

URL: https://leetcode.com/problems/3sum-smaller/

```python
class Solution(object):
    def threeSumSmaller(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        if len(nums) == 0 or len(nums) == 2 or len(nums) == 1:
            return len([])
        else:
            triplet_list = []
            sorted_nums = sorted(nums)
            for i in range(0, len(nums) - 2):
                start = i + 1
                end = len(nums) - 1
                while start < end:
                    curr_sum = sorted_nums[i] + sorted_nums[start] + sorted_nums[end]
                    if curr_sum == target:
                        end -= 1
                    elif curr_sum < target:
                        triplet = (sorted_nums[i], sorted_nums[start], sorted_nums[end])
                        triplet_list.append(triplet)
                        start += 1
                    elif curr_sum > target:
                        end -= 1
            print(triplet_list)
            #return len([list(entries) for entries in set(triplet_list)])
            return len(triplet_list)


if __name__ == "__main__":
    soln = Solution()
    print(soln.threeSumSmaller([3,1,0,-2], 4))
```

3 Sum Closest

Given an array S of n integers, find three integers in S such that the sum is closest to a given number, target. Return the sum of the three integers. You may assume that each input would have exactly one solution.

```
For example, given array S = {-1 2 1 -4}, and target = 1.

The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).
```

URL: https://leetcode.com/problems/3sum-closest/

```python
import sys
class Solution(object):
    def threeSumClosest(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        if len(nums) in [0,1,2]:
            return 0
        else:
            min_diff = sys.maxsize
            result = 0
            sorted_nums = sorted(nums)
            for i in range(len(nums)):
                start = i + 1
                end = len(nums) - 1
                while start < end:
                    curr_sum = sorted_nums[i] + sorted_nums[star
t] + sorted_nums[end]
                    diff = abs(curr_sum - target)
                    if diff == 0:
                        return curr_sum
                    if diff < min_diff:
                        min_diff = diff
                        result = curr_sum
                    if curr_sum <= target:
                        start += 1
                    else:
                        end -= 1
            return result

if __name__ == "__main__":

    soln = Solution()
    print(soln.threeSumClosest([-1, 2, 1, -4], 1))
    print(soln.threeSumClosest([-1, 2, 1, -4], 3))
```

3 Sum

Given an array S of n integers, are there elements a, b, c in S 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.

For example, given array S = [-1, 0, 1, 2, -1, -4],

A solution set is: [[-1, 0, 1], [-1, -1, 2]]

URL: https://leetcode.com/problems/3sum/

```python
class Solution(object):
    def threeSum(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        if len(nums) == 0 or len(nums) == 2 or len(nums) == 1:
            return []
        else:
            sum_zero_list = []
            sorted_nums = sorted(nums)
            for i in range(0, len(nums) - 2):
                start = i + 1
                end = len(nums) - 1
                while start < end:
                    curr_sum = sorted_nums[i] + sorted_nums[start] + sorted_nums[end]
                    if curr_sum == 0:
                        zero_triplet = (sorted_nums[i], sorted_nums[start], sorted_nums[end])
                        sum_zero_list.append(zero_triplet)
                        start += 1
                        end -= 1
                    elif curr_sum < 0:
                        start += 1
                    elif curr_sum > 0:
                        end -= 1

            return [list(entries) for entries in set(sum_zero_list)]


if __name__ == "__main__":
    soln = Solution()
    print(soln.threeSum([-1, 0, 1, 2, -1, -4]))
```

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.

Example: Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9, return [0, 1].

URL: https://leetcode.com/problems/two-sum/

```python
class Solution(object):
    def twoSum(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: List[int]
        """
        dict = {}
        for i in range(len(nums)):
            x = nums[i]
            if target-x in dict:
                return (dict[target-x], i)
            dict[x] = i
```

Plus One

Given a non-negative number represented as an array of digits, plus one to the number.

The digits are stored such that the most significant digit is at the head of the list.

URL: https://leetcode.com/problems/plus-one/

```python
class Solution(object):
    def plusOne(self, digits):
        """
        :type digits: List[int]
        :rtype: List[int]
        """
        if len(digits) <= 0:
            return [0]
        else:
            carry = 1
            i = len(digits)-1
            running_sum = 0
            new_digits = []
            while i >= 0:
                running_sum = digits[i] + carry
                if running_sum >= 10:
                    carry = 1
                else:
                    carry = 0
                new_digits.append(running_sum % 10)
                i -= 1

            if carry == 1:
                new_digits.append(1)
                return new_digits[::-1]
            else:
                return new_digits[::-1]
```

Best Time to Buy and Sell Stock

Say you have an array for which the ith element is the price of a given stock on day i.

If you were only permitted to complete at most one transaction (ie, buy one and sell one share of the stock), design an algorithm to find the maximum profit.

Example 1: Input: [7, 1, 5, 3, 6, 4] Output: 5

max. difference = 6-1 = 5 (not 7-1 = 6, as selling price needs to be larger than buying price) Example 2: Input: [7, 6, 4, 3, 1] Output: 0

In this case, no transaction is done, i.e. max profit = 0.

URL: https://leetcode.com/problems/best-time-to-buy-and-sell-stock/

```python
class Solution(object):
    def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        if len(prices) == 0:
            return 0
        else:
            max_profit = 0
            min_price = prices[0]
            for i in range(len(prices)):
                profit = prices[i] - min_price
                max_profit = max(profit, max_profit)
                min_price = min(min_price, prices[i])

            return max_profit
```

Shortest Word Distance

Given a list of words and two words word1 and word2, return the shortest distance between these two words in the list.

For example, Assume that words = ["practice", "makes", "perfect", "coding", "makes"].

Given word1 = "coding", word2 = "practice", return 3. Given word1 = "makes", word2 = "coding", return 1.

Note: You may assume that word1 does not equal to word2, and word1 and word2 are both in the list.

URL: https://leetcode.com/problems/shortest-word-distance/

```python
import sys
class Solution(object):
    def shortestDistance(self, words, word1, word2):
        """
        :type words: List[str]
        :type word1: str
        :type word2: str
        :rtype: int
        """

        word2_positions = []
        word1_positions = []

        for i in range(len(words)):
            if word1 == words[i]:
                word1_positions.append(i)
            if word2 == words[i]:
                word2_positions.append(i)

        min_dist = sys.maxint

        for pos1 in word1_positions:
            for pos2 in word2_positions:
                if abs(pos1 - pos2) < min_dist:
                    min_dist = abs(pos1 - pos2)

        return min_dist
```

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.

For example, given nums = [0, 1, 0, 3, 12], after calling your function, nums should be [1, 3, 12, 0, 0].

Note: You must do this in-place without making a copy of the array. Minimize the total number of operations.

URL:https://leetcode.com/problems/move-zeroes/

```python
class Solution(object):
    def moveZeroes(self, nums):
        """
        :type nums: List[int]
        :rtype: void Do not return anything, modify nums in-plac
e instead.
        """

        i = 0
        j = 0
        while j < len(nums):
            if nums[j] == 0:
                j += 1
            else:
                nums[i] = nums[j]
                i += 1
                j += 1

        while i < len(nums):
            nums[i] = 0
            i += 1
```

Contains Duplicate II

Given an array of integers and an integer k, find out whether there are two distinct indices i and j in the array such that nums[i] = nums[j] and the difference between i and j is at most k.

URL: https://leetcode.com/problems/contains-duplicate-ii/

```python
class Solution(object):
    def containsNearbyDuplicate(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: bool
        """
        if not nums:
            return False
        elif len(nums) == 1:
            return False
        elif len(nums) == 2:
            if nums[0] != nums[1]:
                return False
            else:
                if nums[0] == nums[1] and k >= 1:
                    return True
                else:
                    return False
        else:
            index_dict = {}
            for i in range(len(nums)):
                if nums[i] in index_dict:
                    prev_index = index_dict[nums[i]]
                    if i - prev_index <= k:
                        return True
                index_dict[nums[i]] = i
            return False
```

Majority Element

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

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

URL: https://leetcode.com/problems/majority-element/

```python
class Solution(object):
    def majorityElement(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        candidate = self.get_candidate(nums)
        candidate_count = 0
        if candidate != None:
            for entries in nums:
                if entries == candidate:
                    candidate_count += 1
            if candidate_count >= len(nums)//2:
                return candidate
            else:
                return None
        else:
            return None


    def get_candidate(self, nums):
        count = 0
        candidate = None
        for entries in nums:
            if count == 0:
                candidate = entries
                count = 1
            else:
                if candidate == entries:
                    count += 1
                else:
                    count -= 1
        if count > 0:
            return candidate
        else:
            return None
```

Remove Duplicates from Sorted Array

Given a sorted array, remove the duplicates in place such that each element appear only once and return the new length.

Do not allocate extra space for another array, you must do this in place with constant memory.

For example, Given input array nums = [1,1,2],

Your function should return length = 2, with the first two elements of nums being 1 and 2 respectively. It doesn't matter what you leave beyond the new length.

URL: https://leetcode.com/problems/remove-duplicates-from-sorted-array/

```python
class Solution(object):
    def removeDuplicates(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if len(nums) < 2:
            return len(nums)
        else:
            j = 0
            i = 1
            while i < len(nums):
                if nums[j] == nums[i]:
                    i += 1
                else:
                    j += 1
                    nums[j] = nums[i]
                    i += 1
            return j+1
```

Nested List Weight Sum

Given a nested list of integers, return the sum of all integers in the list weighted by their depth.

Each element is either an integer, or a list -- whose elements may also be integers or other lists.

Example 1: Given the list [[1,1],2,[1,1]], return 10. (four 1's at depth 2, one 2 at depth 1)

Example 2: Given the list [1,[4,[6]]], return 27. (one 1 at depth 1, one 4 at depth 2, and one 6 at depth 3; 1 + 4*2 + 6*3 = 27)

URL: https://leetcode.com/problems/nested-list-weight-sum/

```python
# """
# This is the interface that allows for creating nested lists.
# You should not implement it, or speculate about its implementation
# """
#class NestedInteger(object):
#    def isInteger(self):
#        """
#        @return True if this NestedInteger holds a single integer, rather than a nested list.
#        :rtype bool
#        """
#
#    def getInteger(self):
#        """
#        @return the single integer that this NestedInteger holds, if it holds a single integer
#        Return None if this NestedInteger holds a nested list
#        :rtype int
#        """
#
#    def getList(self):
```

```
#        """
#        @return the nested list that this NestedInteger holds,
if it holds a nested list
#        Return None if this NestedInteger holds a single intege
r
#        :rtype List[NestedInteger]
#        """


class Solution(object):
    def depthSum(self, nestedList):
        """
        :type nestedList: List[NestedInteger]
        :rtype: int
        """
        return self.depthSum_helper(nestedList, 1)


    def depthSum_helper(self, nested_list, depth):
        if len(nested_list) == 0 or nested_list == None:
            return 0
        else:
            sum = 0
            for entries in nested_list:
                if entries.isInteger():
                    sum += entries.getInteger()*depth
                else:
                    sum += self.depthSum_helper(entries.getList(
), depth + 1)

            return sum
```

Nested List Weighted Sum II

Given a nested list of integers, return the sum of all integers in the list weighted by their depth.

Each element is either an integer, or a list -- whose elements may also be integers or other lists.

Different from the previous question where weight is increasing from root to leaf, now the weight is defined from bottom up. i.e., the leaf level integers have weight 1, and the root level integers have the largest weight.

Example 1: Given the list [[1,1],2,[1,1]], return 8. (four 1's at depth 1, one 2 at depth 2)

Example 2: Given the list [1,[4,[6]]], return 17. (one 1 at depth 3, one 4 at depth 2, and one 6 at depth 1; 1*3 + 4*2 + 6*1 = 17)

URL: https://leetcode.com/problems/nested-list-weight-sum-ii/

Remove Element

Given an array and a value, remove all instances of that value in place and return the new length.

Do not allocate extra space for another array, you must do this in place with constant memory.

The order of elements can be changed. It doesn't matter what you leave beyond the new length.

Example: Given input array nums = [3,2,2,3], val = 3

Your function should return length = 2, with the first two elements of nums being 2.

URL: https://leetcode.com/problems/remove-element/

```python
class Solution(object):
    def removeElement(self, nums, val):
        """
        :type nums: List[int]
        :type val: int
        :rtype: int
        """
        if val == []:
            return 0
        else:
            i = 0
            j = 0
            while j < len(nums):
                if nums[j] == val:
                    j += 1
                else:
                    nums[i] = nums[j]
                    i += 1
                    j += 1

            return len(nums[0:i])
```

Intersection of Two Arrays II

Given two arrays, write a function to compute their intersection.

Example: Given nums1 = [1, 2, 2, 1], nums2 = [2, 2], return [2, 2].

Note: Each element in the result should appear as many times as it shows in both arrays. The result can be in any order. Follow up: What if the given array is already sorted? How would you optimize your algorithm? What if nums1's size is small compared to nums2's size? Which algorithm is better? What if elements of nums2 are stored on disk, and the memory is limited such that you cannot load all elements into the memory at once?

URL: https://leetcode.com/problems/intersection-of-two-arrays-ii/

```python
class Solution(object):
    def intersect(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: List[int]
        """
        sorted_nums1 = sorted(nums1)
        sorted_nums2 = sorted(nums2)

        i = 0
        j = 0
        intersect_list = []

        while i < len(sorted_nums1) and j < len(sorted_nums2):
            if sorted_nums1[i] < sorted_nums2[j]:
                i += 1
            elif sorted_nums2[j] < sorted_nums1[i]:
                j += 1
            else:
                intersect_list.append(sorted_nums1[i])
                i += 1
                j += 1

        return intersect_list
```

Merge Sorted Arrays

Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.

Note: You may assume that nums1 has enough space (size that is greater or equal to m + n) to hold additional elements from nums2. The number of elements initialized in nums1 and nums2 are m and n respectively.

URL: https://leetcode.com/problems/merge-sorted-array/

```python
class Solution(object):
    def merge(self, nums1, m, nums2, n):
        """
        :type nums1: List[int]
        :type m: int
        :type nums2: List[int]
        :type n: int
        :rtype: void Do not return anything, modify nums1 in-pla
ce instead.
        """
        last1 = m - 1
        last2 = n - 1
        last = m + n - 1

        while last1 >= 0 and last2 >= 0:
            if nums1[last1] > nums2[last2]:
                nums1[last] = nums1[last1]
                last1 -= 1
                last -= 1
            else:
                nums1[last] = nums2[last2]
                last2 -= 1
                last -= 1

        while last2 >= 0:
            nums1[last] = nums2[last2]
            last -= 1
            last2 -= 1
```

Reverse Vowels of a String

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

Example 1: Given s = "hello", return "holle".

Example 2: Given s = "leetcode", return "leotcede".

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

URL: https://leetcode.com/problems/reverse-vowels-of-a-string/

```python
class Solution(object):
    def __init__(self):
        self.__vowels = {"a" : True, "e" : True, "i" : True, "o"
 : True, "u" : True, "A" : True, "E" : True, "I" : True, "O" : T
rue, "U" :                 True,}


    def reverseVowels(self, s):
        """
        :type s: str
        :rtype: str
        """
        if s == None or s == "" or len(s) == 1:
            return s
        else:
            i=0
            j = len(s) - 1
            s = list(s)

            while i < j:
                if s[i] not in self.__vowels:
                    i += 1
                    continue
                if s[j] not in self.__vowels:
                    j -= 1
                    continue
                s[i], s[j] = s[j], s[i]
                i += 1
                j -= 1
            return "".join(s)
```

Intersection of Two Arrays

Given two arrays, write a function to compute their intersection.

Example: Given nums1 = [1, 2, 2, 1], nums2 = [2, 2], return [2].

Note: Each element in the result must be unique. The result can be in any order.

URL: https://leetcode.com/problems/intersection-of-two-arrays/

```python
class Solution(object):
    def intersection(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: List[int]
        """
        nums1 = sorted(nums1)
        nums2 = sorted(nums2)
        intersection = {}
        i = 0
        j = 0
        while i < len(nums1) and j < len(nums2):
            if nums1[i] < nums2[j]:
                i += 1
            elif nums2[j] < nums1[i]:
                j += 1
            else:
                intersection[nums1[i]] = nums1[i]
                i += 1
                j += 1

        return intersection.keys()
```

Container With Most Water

Givennnon-negative integersa1,a2, ...,an, where each represents a point at coordinate (i,ai).nvertical lines are drawn such that the two endpoints of lineiis at (i,ai) and (i, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.

Note: You may not slant the container and n is at least 2.

URL: https://leetcode.com/problems/container-with-most-water/

```python
class Solution(object):
    def maxArea(self, height):
        """
        :type height: List[int]
        :rtype: int
        """
        max_area = 0
        i = 0
        j = len(height) - 1
        while i<j:
            max_area = max(max_area, min(height[i], height[j])*(
j-i))

            if height[i] < height[j]:
                i += 1
            else:
                j -= 1

        return max_area
```

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

Solve it **without division** and in O(n).

For example, given `[1,2,3,4]` , return `[24,12,8,6]` .

Follow up:

Could you solve it with constant space complexity? (Note: The output array **does not** count as extra space for the purpose of space complexity analysis.)

URL: https://leetcode.com/problems/product-of-array-except-self/

```python
class Solution(object):
    def productExceptSelf(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]
        """
        before = [1]*len(nums)
        after = [1]*len(nums)
        product = [0]*len(nums)

        for i in range(1, len(nums)):
            before[i] = before[i-1]*nums[i-1]

        for i in range(len(nums)-2, -1, -1):
            after[i] = after[i+1]*nums[i+1]

        for i in range(0, len(nums)):
            product[i] = before[i]*after[i]

        return product
```

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

For example,
Given `[0,1,0,2,1,0,1,3,2,1,2,1]`, return `6`.

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

URL: https://leetcode.com/problems/trapping-rain-water/

```python
class Solution(object):
    def trap(self, height):
        """
        :type height: List[int]
        :rtype: int
        """
        maxseenright = 0
        maxseenright_arr = [0]*len(height)
        maxseenleft = 0
        rainwater = 0

        for i in range(len(height) - 1, -1, -1):
            if height[i] > maxseenright:
                maxseenright_arr[i] = height[i]
                maxseenright = height[i]
            else:
                maxseenright_arr[i] = maxseenright


        for i in range(0, len(height)):
            rainwater = rainwater + max(min(maxseenright_arr[i],
 maxseenleft) - height[i],0)
            if height[i] > maxseenleft:
                maxseenleft = height[i]

        return rainwater
```

Find the contiguous subarray within an array (containing at least one number) which has the largest sum.

For example, given the array `[-2,1,-3,4,-1,2,1,-5,4]`, the contiguous subarray `[4,-1,2,1]` has the largest sum = `6`.

URL: https://leetcode.com/problems/maximum-subarray/

```python
import sys
class Solution(object):
    def maxSubArray(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if nums == []:
            return 0
        elif len(nums) == 1:
            return nums[0]
        elif len(nums) == 2:
            return max(nums[0], nums[1], nums[0]+nums[1])
        else:
            all_neg = True
            for entries in nums:
                if entries >= 0:
                    all_neg = False
            if all_neg == False:
                curr_sum = 0
                max_sum = - sys.maxsize - 1
                for i in range(len(nums)):
                    curr_sum += nums[i]
                    if curr_sum < 0:
                        curr_sum = 0
                    if curr_sum > max_sum:
                        max_sum = curr_sum
                return max_sum
            else:
                return max(nums)
```

Say you have an array for which the ith element is the price of a given stock on day i.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times). However, you may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

URL: https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/

```python
class Solution(object):
    def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        if prices == []:
            return 0
        else:
            profit = 0
            for i in range(1, len(prices)):
                curr_profit = prices[i] - prices[i-1]
                if curr_profit > 0:
                    profit += curr_profit
            return profit
```

Suppose a sorted array 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`).

Find the minimum element.

You may assume no duplicate exists in the array.

URL: https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/

```python
class Solution(object):
    def findMin(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        start = 0
        end = len(nums) - 1

        while start < end:
            mid = start + (end - start)// 2
            if nums[mid] >= nums[end]:
                start = mid + 1
            else:
                end = mid
        return nums[start]
```

Given num Rows, generate the firstnum Rows of Pascal's triangle.

For example, givennumRows= 5,
Return

```
[
     [1],
    [1,1],
   [1,2,1],
  [1,3,3,1],
 [1,4,6,4,1]
]
```

URL: https://leetcode.com/problems/pascals-triangle/

```python
class Solution(object):
    def generate(self, numRows):
        """
        :type numRows: int
        :rtype: List[List[int]]
        """
        if numRows <= 0:
            return []

        result = []
        pre = []

        pre.append(1)
        result.append(pre)

        for i in range(0, numRows-1):
            curr = []
            curr.append(1)
            for j in range(0, len(pre)-1):
                curr.append(pre[j]+pre[j+1])
            curr.append(1)
            result.append(curr)
            pre = curr

        return result
```

Given an index k, return the kth row of the Pascal's triangle.

For example, given k= 3,
Return `[1,3,3,1]` .

Note:

Could you optimize your algorithm to use only O(k) extra space?

URL: https://leetcode.com/problems/pascals-triangle-ii/

```python
class Solution(object):
    def getRow(self, rowIndex):
        """
        :type rowIndex: int
        :rtype: List[int]
        """
        if rowIndex < 0:
            return []
        elif rowIndex == 0:
            return [1]
        else:
            pre = []
            pre.append(1)
            for i in range(1, rowIndex+1):
                curr = []
                curr.append(1)
                for j in range(0, len(pre) - 1):
                    curr.append(pre[j]+pre[j+1])
                curr.append(1)
                pre = curr

            return pre
```

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

For example, given `[0,1,2,4,5,7]`, return `["0->2","4->5","7"].`

URL: https://leetcode.com/problems/summary-ranges/

```python
class Solution(object):
    def summaryRanges(self, nums):
        """
        :type nums: List[int]
        :rtype: List[str]
        """
        if nums == []:
            return nums
        elif len(nums) == 1:
            return [str(nums[0])]
        else:
            start = nums[0]
            end = nums[0]
            res = []
            for i in range(1, len(nums)):
                if nums[i] - nums[i-1] == 1:
                    end = nums[i]
                else:
                    res.append(self.to_str(start, end))
                    start = end = nums[i]

            res.append(self.to_str(start, end))

            return res

    def to_str(self, start, end):
        if start == end:
            return str(start)
        else:
            return str(start)+"->"+str(end)
```

Given an array containing n distinct numbers taken from `0, 1, 2, ..., n` , find the one that is missing from the array.

For example,
Given nums= `[0, 1, 3]` return `2` .

Note:
Your algorithm should run in linear runtime complexity. Could you implement it using only constant extra space complexity?

URL: https://leetcode.com/problems/missing-number/

```python
class Solution(object):
    def missingNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return None
        else:
            xor_prod = 0
            xor_prod_index = 0

            for i in range(len(nums)+1):
                xor_prod_index ^= i

            for i in range(len(nums)):
                xor_prod ^= nums[i]

            return xor_prod ^ xor_prod_index
```

Valid Anagram

Given two strings s and t, write a function to determine if t is an anagram of s.

For example, s = "anagram", t = "nagaram", return true. s = "rat", t = "car", return false.

Note: You may assume the string contains only lowercase alphabets.

URL: https://leetcode.com/problems/valid-anagram/

```python
class Solution(object):
    def isAnagram(self, s, t):
        """
        :type s: str
        :type t: str
        :rtype: bool
        """
        if len(s) != len(t):
            return False
        elif sorted(s) == sorted(t):
            return True
        else:
            return False
```

Valid Palindrome

Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

For example, "A man, a plan, a canal: Panama" is a palindrome. "race a car" is not a palindrome.

Note: Have you consider that the string might be empty? This is a good question to ask during an interview.

For the purpose of this problem, we define empty string as valid palindrome.

URL: https://leetcode.com/problems/valid-palindrome/

```python
import re
class Solution:
    # @param {string} s
    # @return {boolean}
    def isPalindrome(self, s):
        if len(s) == 0:
            return True
        else:
            start = 0
            s = s.lower()
            newS = re.sub(r"[^a-zA-Z0-9]","",s)
            end = len(newS)-1
            while start < end:
                if newS[start] == newS[end]:
                    start = start + 1
                    end = end - 1
                else:
                    return False
            return True
```

Word Pattern

Given a pattern and a string str, find if str follows the same pattern.

Here follow means a full match, such that there is a bijection between a letter in pattern and a non-empty word in str.

Examples: pattern = "abba", str = "dog cat cat dog" should return true. pattern = "abba", str = "dog cat cat fish" should return false. pattern = "aaaa", str = "dog cat cat dog" should return false. pattern = "abba", str = "dog dog dog dog" should return false. Notes: You may assume pattern contains only lowercase letters, and str contains lowercase letters separated by a single space.

URL: https://leetcode.com/problems/word-pattern/

```python
class Solution(object):
    def wordPattern(self, pattern, str):
        """
        :type pattern: str
        :type str: str
        :rtype: bool
        """
        if pattern == None or str == None:
            return False
        else:
            len_str = len(str.split(" "))
            len_pattern = len(pattern)
            if len_str != len_pattern:
                return False
            str = str.split(" ")
            lookup = {}
            for i in range(0, len(pattern)):
                s = str[i]
                p = pattern[i]
                if p in lookup:
                    if lookup[p] != s:
                        return False
                else:
                    if s in lookup.values():
                        return False
                    lookup[p] = s
            return True

if __name__ == "__main__":

    pattern = "abba"
    str = "dog cat cat dog"
    soln = Solution()
    print(soln.wordPattern(pattern, str))
```

Valid Parentheses

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not.

URL: https://leetcode.com/problems/valid-parentheses/

```
class Solution:
    # @param {string} s
    # @return {boolean}
    def isValid(self, s):
        if s == []:
            return False
        else:
            stack = []
            balanced = True
            index = 0
            while index < len(s) and balanced:
                symbol = s[index]
                if symbol in "({[":
                    stack.append(symbol)
                else:
                    if stack == []:
                        balanced = False
                    else:
                        top = stack.pop()
                        if not self.matches(top,symbol):
                            balanced = False
                index = index + 1

            if balanced and stack == []:
                return True
            else:
                return False


    def matches(self,open,close):
        openings = "({["
        closings = ")}]"

        return openings.index(open) == closings.index(close)
```

Isomorphic Strings

Given two strings s and t, determine if they are isomorphic.

Two strings are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character but a character may map to itself.

For example, Given "egg", "add", return true.

Given "foo", "bar", return false.

Given "paper", "title", return true.

Note: You may assume both s and t have the same length.

URL: https://leetcode.com/problems/isomorphic-strings/

```python
class Solution(object):
    def isIsomorphic(self, s, t):
        """
        :type s: str
        :type t: str
        :rtype: bool
        """
        if s == None or t == None:
            return False
        elif s == "" and t == "":
            return True
        else:
            if len(s) != len(t):
                return False

            lookup = {}
            for i in range(0, len(s)):
                c1 = s[i]
                c2 = t[i]

                if c1 in lookup:
                    if lookup[c1] != c2:
                        return False
                else:
                    if c2 in lookup.values():
                        return False
                    lookup[c1] = c2

            return True
```

Reverse String

Write a function that takes a string as input and returns the string reversed.

Example: Given s = "hello", return "olleh".

URL: https://leetcode.com/problems/reverse-string/

```python
class Solution(object):
    def reverseString(self, s):
        """
        :type s: str
        :rtype: str
        """

        current_str = [char for char in s]

        i = 0
        j = len(s) - 1

        while i < j:
            temp = current_str[i]
            current_str[i] = current_str[j]
            current_str[j] = temp
            j -= 1
            i += 1

        return "".join(current_str)
```

Sum of Two Integers

Calculate the sum of two integers a and b, but you are not allowed to use the operator + and -.

Example: Given a = 1 and b = 2, return 3.

URL: https://leetcode.com/problems/sum-of-two-integers/

```python
class Solution(object):
    def getSum(self, a, b):
        """
        :type a: int
        :type b: int
        :rtype: int
        """
        if b == 0:
            return a
        sum = a ^ b
        carry = (a & b) << 1
        return self.getSum(sum, carry)
```

Single Number

Given an array of integers, every element appears twice except for one. Find that single one.

Note: Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

URL: https://leetcode.com/problems/single-number/

```python
class Solution(object):
    def singleNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if len(nums) == 0:
            return None
        elif len(nums) == 1:
            return nums[0]
        else:
            xor_prod = 0
            for entries in nums:
                xor_prod ^= entries

            return xor_prod
```

Reverse Integer

Reverse digits of an integer.

Example1: x = 123, return 321 Example2: x = -123, return -321

click to show spoilers.

Have you thought about this? Here are some good questions to ask before coding. Bonus points for you if you have already thought through this!

If the integer's last digit is 0, what should the output be? ie, cases such as 10, 100.

Did you notice that the reversed integer might overflow? Assume the input is a 32-bit integer, then the reverse of 1000000003 overflows. How should you handle such cases?

For the purpose of this problem, assume that your function returns 0 when the reversed integer overflows.

URL: https://leetcode.com/problems/reverse-integer/

```
import sys
class Solution(object):
    def reverse(self, x):
        """
        :type x: int
        :rtype: int
        """
        if x < 0:
            return -self.reverse(-x)

        result = 0
        while x:
            result = result * 10 + x % 10
            x /= 10
        return result if result <= 0x7fffffff else 0
```

Palindrome Number

Determine whether an integer is a palindrome. Do this without extra space.

click to show spoilers.

Some hints: Could negative integers be palindromes? (ie, -1)

If you are thinking of converting the integer to string, note the restriction of using extra space.

You could also try reversing an integer. However, if you have solved the problem "Reverse Integer", you know that the reversed integer might overflow. How would you handle such case?

There is a more generic way of solving this problem.

URL: https://leetcode.com/problems/palindrome-number/

```python
class Solution(object):
    def isPalindrome(self, x):
        """
        :type x: int
        :rtype: bool
        """
        if x < 0:
            return False

        rev = 0
        copy = x
        while copy != 0:
            rev = rev*10 + copy%10
            copy = copy/10
        if rev == x:
            return True
        else:
            return False
```

Pow(x,n)

Implement pow(x, n).

```python
class Solution(object):
    def myPow(self, x, n):
        """
        :type x: float
        :type n: int
        :rtype: float
        """
        if n < 0:
            return 1/self.power(x, -n)
        else:
            return self.power(x, n)

    def power(self, x, n):
        if n == 0:
            return 1

        v = self.power(x, n//2)

        if n % 2 == 0:
            return v * v
        else:
            return v * v * x
```

Subsets

Given a set of distinct integers, nums, return all possible subsets.

Note: The solution set must not contain duplicate subsets.

For example, If nums = [1,2,3], a solution is:

[[3], [1], [2], [1,2,3], [1,3], [2,3], [1,2], []]

URL: https://leetcode.com/problems/subsets/

Solution1:

```python
class Solution(object):
    def subsets(self, S):
        def dfs(depth, start, valuelist):
            res.append(valuelist)
            if depth == len(S): return
            for i in range(start, len(S)):
                dfs(depth+1, i+1, valuelist+[S[i]])
        S.sort()
        res = []
        dfs(0, 0, [])
        return res
```

Solution2:

```python
class Solution(object):
    def subsets(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        n = 1 << len(nums)
        result = []
        for i in range(0, n):
            subset = self.convert(i, nums)
            result.append(subset)

        return result


    def convert(self, i, nums):
        k = i
        index = 0
        res = []
        while k > 0:
            if (k & 1) == 1:
                res.append(nums[index])

            k >>= 1
            index += 1
        return res
```

Subsets II

Given a collection of integers that might contain duplicates, nums, return all possible subsets.

Note: The solution set must not contain duplicate subsets.

For example, If nums = [1,2,2], a solution is:

[[2], [1], [1,2,2], [2,2], [1,2], []]

URL: https://leetcode.com/problems/subsets-ii/

```python
class Solution(object):
    def subsetsWithDup(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        n = 1 << len(nums)
        result = []
        for i in range(0, n):
            subset = self.convert(i, nums)
            result.append(tuple(sorted(subset)))

        result = set(result)
        return [list(entries) for entries in result]

    def convert(self, i, nums):
        k = i
        index = 0
        res = []
        while k > 0:
            if (k & 1) == 1:
                res.append(nums[index])

            k >>= 1
            index += 1
        return res
```
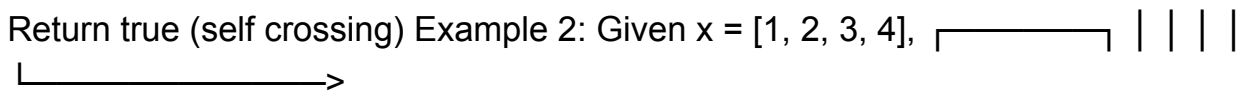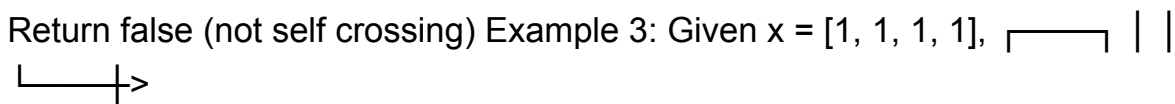
Self Crossing

You are given an array x of n positive numbers. You start at point (0,0) and moves x[0] metres to the north, then x[1] metres to the west, x[2] metres to the south, x[3] metres to the east and so on. In other words, after each move your direction changes counter-clockwise.

Write a one-pass algorithm with O(1) extra space to determine, if your path crosses itself, or not.

Example 1: Given x = [2, 1, 1, 2], ┌─────┐ │ │ └───┼──> │

Return true (self crossing) Example 2: Given x = [1, 2, 3, 4], ┌───────┐ │ │ │ │ └──────────────>

Return false (not self crossing) Example 3: Given x = [1, 1, 1, 1], ┌─────┐ │ │ └────┼>

Return true (self crossing)

URL: https://leetcode.com/problems/self-crossing/

```python
class Solution(object):
    def isSelfCrossing(self, x):
        """
        :type x: List[int]
        :rtype: bool
        """
        if x == None or len(x) <= 3:
            return False
        else:
            for i in range(3, len(x)):
                if (x[i-3] >= x[i-1]) and (x[i-2] <= x[i]):
                    return True
                if (i >= 4) and (x[i-4] + x[i] >= x[i-2]) and (x
[i-3] == x[i-1]):
                    return True
                if (i>=5) and (x[i-5] <= x[i-3]) and (x[i-4] <=
x[i-2]) and (x[i-1] <= x[i-3]) and (x[i-1] >= x[i-3] - x[i-5]) a
nd (x[i] >= x[i-2] - x[i-4]) and (x[i] <= x[i-2]):
                    return True

            return False
```

Paint Fence

There is a fence with n posts, each post can be painted with one of the k colors.

You have to paint all the posts such that no more than two adjacent fence posts have the same color.

Return the total number of ways you can paint the fence.

Note: n and k are non-negative integers.

URL: https://leetcode.com/problems/paint-fence/

```python
class Solution(object):
    def numWays(self, n, k):
        """
        :type n: int
        :type k: int
        :rtype: int
        """
        dp = [0, k, k*k, 0]
        if n <= 2:
            return dp[n]
        for i in range(2, n):
            dp[3] = (k-1)*(dp[1] + dp[2])
            dp[1] = dp[2]
            dp[2] = dp[3]

        return dp[3]
```

Bulb Switcher

There are n bulbs that are initially off. You first turn on all the bulbs. Then, you turn off every second bulb. On the third round, you toggle every third bulb (turning on if it's off or turning off if it's on). For the ith round, you toggle every i bulb. For the nth round, you only toggle the last bulb. Find how many bulbs are on after n rounds.

Example:

Given n = 3.

At first, the three bulbs are [off, off, off]. After first round, the three bulbs are [on, on, on]. After second round, the three bulbs are [on, off, on]. After third round, the three bulbs are [on, off, off].

So you should return 1, because there is only one bulb is on.

URL: https://leetcode.com/problems/bulb-switcher/

```python
import math
class Solution(object):
    def bulbSwitch(self, n):
        """
        :type n: int
        :rtype: int
        """
        return int(math.sqrt(n))
```

Nim Game

You are playing the following Nim Game with your friend: There is a heap of stones on the table, each time one of you take turns to remove 1 to 3 stones. The one who removes the last stone will be the winner. You will take the first turn to remove the stones.

Both of you are very clever and have optimal strategies for the game. Write a function to determine whether you can win the game given the number of stones in the heap.

For example, if there are 4 stones in the heap, then you will never win the game: no matter 1, 2, or 3 stones you remove, the last stone will always be removed by your friend.

Hint:

If there are 5 stones in the heap, could you figure out a way to remove the stones such that you will always be the winner?

URL: https://leetcode.com/problems/nim-game/

```python
class Solution(object):
    def canWinNim(self, n):
        """
        :type n: int
        :rtype: bool
        """
        return n%4 != 0
```

Rotate Image

You are given an n x n 2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

Follow up: Could you do this in-place?

URL: https://leetcode.com/problems/rotate-image/

```python
class Solution(object):
    def rotate(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: void Do not return anything, modify matrix in-place instead.
        """
        if matrix == None or matrix == []:
            pass
        else:
            n = len(matrix)
            for layer in range(0, n//2):
                first = layer
                last = n - 1 - layer
                for i in range(first, last):
                    offset = i - first
                    top = matrix[first][i]
                    matrix[first][i] = matrix[last - offset][first]
                    matrix[last - offset][first] = matrix[last][last - offset]
                    matrix[last][last - offset] = matrix[i][last]
                    matrix[i][last] = top
```

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.

click to show follow up.

Follow up: Did you use extra space? 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?

URL: https://leetcode.com/problems/set-matrix-zeroes/

```python
class Solution(object):
    def setZeroes(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: void Do not return anything, modify matrix in-pl
ace instead.
        """
        if matrix == None or len(matrix) == 0:
            pass
        elif len(matrix) == 1 and len(matrix[0]) == 1:
            pass
        else:
            rows_with_0 = [False]*len(matrix)
            cols_with_0 = [False]*len(matrix[0])
            for i in range(len(matrix)):
                for j in range(len(matrix[0])):
                    if matrix[i][j] == 0:
                        rows_with_0[i] = True
                        cols_with_0[j] = True

            for i in range(len(matrix)):
                for j in range(len(matrix[0])):
                    if rows_with_0[i] or cols_with_0[j]:
                        matrix[i][j] = 0
```

Constant Space Solution:

```python
class Solution(object):
    def setZeroes(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: void Do not return anything, modify matrix in-place instead.
        """
        first_row = False
        first_col = False

        for j in range(len(matrix[0])):
            if matrix[0][j] == 0:
                first_row = True

        for i in range(len(matrix)):
            if matrix[i][0] == 0:
                first_col = True


        for i in range(1, len(matrix)):
            for j in range(1, len(matrix[0])):
                if matrix[i][j] == 0:
                    matrix[i][0] = 0
                    matrix[0][j] = 0

        for i in range(1, len(matrix)):
            for j in range(1, len(matrix[0])):
                if matrix[i][0] == 0 or matrix[0][j] == 0:
                    matrix[i][j] = 0

        if first_col:
            for i in range(len(matrix)):
                matrix[i][0] = 0

        if first_row:
            for i in range(len(matrix[0])):
                matrix[0][i] = 0
```

Search a 2D Matrix

Write an efficient algorithm that searches for a value in an m x n matrix. This matrix has the following properties:

Integers in each row are sorted from left to right. The first integer of each row is greater than the last integer of the previous row. For example,

Consider the following matrix:

[[1, 3, 5, 7], [10, 11, 16, 20], [23, 30, 34, 50]] Given target = 3, return true.

```python
class Solution(object):
    def searchMatrix(self, matrix, target):
        """
        :type matrix: List[List[int]]
        :type target: int
        :rtype: bool
        """
        if matrix == []:
            return False
        else:
            no_rows = len(matrix)
            no_cols = len(matrix[0])

            #get the first element and the last element of the m
atrix
            #compare it with the target
            if target < matrix[0][0] or target > matrix[no_rows-
1][no_cols-1]:
                return False

            r = 0
            c = no_cols - 1
            while r < no_rows and c >= 0:
                if target == matrix[r][c]:
                    return True
                elif target > matrix[r][c]:
                    r += 1
                elif target < matrix[r][c]:
                    c -= 1
            return False
```

Search a 2D Matrix II

Write an efficient algorithm that searches for a value in an m x n matrix. This matrix has the following properties:

Integers in each row are sorted in ascending from left to right. Integers in each column are sorted in ascending from top to bottom. For example,

Consider the following matrix:

[[1, 4, 7, 11, 15], [2, 5, 8, 12, 19], [3, 6, 9, 16, 22], [10, 13, 14, 17, 24], [18, 21, 23, 26, 30]] Given target = 5, return true.

Given target = 20, return false.

URL: https://leetcode.com/problems/search-a-2d-matrix-ii/

```python
class Solution(object):
    def searchMatrix(self, matrix, target):
        """
        :type matrix: List[List[int]]
        :type target: int
        :rtype: bool
        """
        if matrix == []:
            return False
        else:
            no_rows = len(matrix)
            no_cols = len(matrix[0])

            if target < matrix[0][0] or target > matrix[no_rows-
1][no_cols-1]:
                return False
            else:
                r = 0
                c = no_cols-1

                while r < no_rows and c >=0:
                    if matrix[r][c] == target:
                        return True
                    elif target > matrix[r][c]:
                        r += 1
                    elif target < matrix[r][c]:
                        c -= 1
                return False
```

Spiral Matrix

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

For example, Given the following matrix:

[[1, 2, 3], [4, 5, 6], [7, 8, 9]] You should return [1,2,3,6,9,8,7,4,5].

URL: https://leetcode.com/problems/spiral-matrix/

```python
class Solution(object):
    def spiralOrder(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: List[int]
        """
        if matrix == None or matrix == []:
            return matrix
        else:
            #no of rows
            m = len(matrix)
            #no of columns
            n = len(matrix[0])
            #starting row
            k = 0
            #starting column
            l = 0

            #spiral order matrix
            spiral = []

            while k < m and l < n:
                #print the first row from the remaining rows
                for i in range(l, n):
                    spiral.append(matrix[k][i])
                k += 1

                #print the last column from the remaining column
```

```
s
            for i in range(k, m):
                spiral.append(matrix[i][n-1])
            n-= 1

            #print the last row from the remaining rows
            if k < m:
                for i in range(n-1, l-1, -1):
                    spiral.append(matrix[m-1][i])

                m -= 1

            #print the first column from the remaining colum
ns
            if l < n:
                for i in range(m-1, k-1, -1):
                    spiral.append(matrix[i][l])

                l += 1

        return spiral
```

Spiral Matrix II

Given an integer n, generate a square matrix filled with elements from 1 to n2 in spiral order.

For example, Given n = 3,

You should return the following matrix: [[1, 2, 3], [8, 9, 4], [7, 6, 5]]

URL: https://leetcode.com/problems/spiral-matrix-ii/

```python
class Solution(object):
    def generateMatrix(self, n):
        """
        :type n: int
        :rtype: List[List[int]]
        """
        if n == 0:
            return []
        elif n == 1:
            return [[1]]
        else:
            #no of rows
            r = n
            #no of columns
            c = n
            #start of row
            k = 0
            #start of column
            l = 0

            #allocate a square matrix with all zeros
            matrix = [[0 for j in range(c)] for i in range(r)]
            #counter for the elements
            count = 1

            while k < r and l < c:
                #fill the first row
                for i in range(l, c):
```

```python
                matrix[k][i] = count
                count += 1

            k += 1

            #fill the last column
            for i in range(k, r):
                matrix[i][c-1] = count
                count += 1

            c -= 1

            #fill the last row
            if k < r:
                for i in range(c-1, l-1, -1):
                    matrix[r-1][i] = count
                    count += 1
                r -= 1

            #fill the first column
            if l < c:
                for i in range(r-1, k-1, -1):
                    matrix[i][l] = count
                    count += 1
                l += 1

        return matrix
```

LRU Cache

Design and implement a data structure for Least Recently Used (LRU) cache. It should support the following operations: `get` and `set` .

`get(key)` - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.

`set(key, value)` - Set or insert the value if the key is not already present. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item.

```python
class LRUCache(object):

    def __init__(self, capacity):
        """
        :type capacity: int
        """
        self.capacity = capacity
        self.cache = OrderedDict()



    def get(self, key):
        """
        :rtype: int
        """
        if key in self.cache:
            value = self.cache.pop(key)
            self.cache[key] = value
            return value
        else:
            return -1



    def set(self, key, value):
        """
        :type key: int
        :type value: int
        :rtype: nothing
        """
        if len(self.cache) >= self.capacity and key not in self.cache:
            self.cache.popitem(last=False)
            self.cache[key] = value
        else:
            if key in self.cache:
                self.cache.pop(key)
                self.cache[key] = value
            else:
                self.cache[key] = value
```