35. Implement Queue Using Stack

```
class MyQueue:

    def __init__(self):
        self.push_stack = []
        self.pop_stack = []

    def push(self, x: int) -> None:
        self.push_stack.append(x)

    def pop(self) -> int:
        if self.empty():
            return
        if len(self.pop_stack):
            return self.pop_stack.pop()
        else:
            while len(self.push_stack):
                self.pop_stack.append(self.push_stack.pop())
        return self.pop_stack.pop()

    def peek(self) -> int:
        if self.empty():
            return
        if len(self.pop_stack):
            return self.pop_stack[-1]
        else:
            while len(self.push_stack):
                self.pop_stack.append(self.push_stack.pop())
        return self.pop_stack[-1]

    def empty(self) -> bool:
        return len(self.push_stack)==False and len(self.pop_stack) ==
False
```

36. Add Two Numbers

- Input: l1 = [2,4,3], l2 = [5,6,4]
- Output: [7,0,8]
- Explanation: 342 + 465 = 807.

```
class Solution:
    def addTwoNumbers(self, l1: Optional[ListNode], l2:
Optional[ListNode]) -> Optional[ListNode]:
        head = None
        temp = None
        carry = 0
        while l1 is not None or l2 is not None:
            sum_value = carry
            if l1 is not None:
                sum_value += l1.val
```

```
            l1 = l1.next
        if l2 is not None:
            sum_value += l2.val
            l2 = l2.next
        node = ListNode(sum_value % 10)
        carry = sum_value // 10
        if temp is None:
            temp = head = node
        else:
            temp.next = node
            temp = temp.next
    if carry > 0:
        temp.next = ListNode(carry)
    return head
```

37. Merge Sorted Array

- Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3
- Output: [1,2,2,3,5,6]
- Explanation: The arrays we are merging are [1,2,3] and [2,5,6].
- The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.

```
# Time O(n+m)
# Space O(n+m)

def mergeArrays(arr1, arr2, n1, n2):
    arr3 = [None] * (n1 + n2)
    i = 0
    j = 0
    k = 0

    while i < n1 and j < n2:
        # Проверяем, является ли текущий элемент первого массива
меньше чем текущий элемент
        # второй массив. Если да, сохраним первый элемент массива и
увеличиваем первый массив
        # индекс. В противном случае сделайте то же самое со вторым
массивом

        if arr1[i] < arr2[j]:
            arr3[k] = arr1[i]
            k = k + 1
            i = i + 1
        else:
            arr3[k] = arr2[j]
            k = k + 1
            j = j + 1

    # Сохраним оставшиеся элементы первого массива
    while i < n1:
```

```
        arr3[k] = arr1[i];
        k = k + 1
        i = i + 1

    # Сохраняем оставшиеся элементы второго массива
    while j < n2:
        arr3[k] = arr2[j];
        k = k + 1
        j = j + 1

    for i in range(n1 + n2):
        print(str(arr3[i]), end = " ")
```

38. Merge Two Sorted Lists

- Input: list1 = [1,2,4], list2 = [1,3,4]
- Output: [1,1,2,3,4,4]

```
# Time O(n)
# Space O(1)

# Объединяет два списка с заголовками h1 и h2.
# Предполагается, что данные h1 меньше, чем
# или равно данным h2.

def mergeUtil(h1, h2):
    # если только один узел в первом списке просто указать голову на
второй список

    if (h1.next == None):
        h1.next = h2
        return h1

    # Инициализируем текущий и следующий указатели обоих списков

    curr1 = h1
    next1 = h1.next
    curr2 = h2
    next2 = h2.next

    while next1 != None and curr2 != None:
        # если curr2 находится между curr1 и next1, то делаем
curr1.curr2.next1

        if curr2.data >= curr1.data and curr2.data <= next1.data:
            next2 = curr2.next
            curr1.next = curr2
            curr2.next = next1

            # теперь пусть curr1 и curr2 указывают на свои
непосредственные указатели next
```

```
            curr1 = curr2
            curr2 = next2
        else:
            # если больше узлов в первом списке
            if (next1.next):
                next1 = next1.next
                curr1 = curr1.next
            # иначе указать последний узел первого списка к остальным
узлам второго списка
            else:
                next1.next = curr2
                return h1

    return h1


# Объединяет два заданных списка на месте. Эта функция в основном
сравнивает голову узлы и вызовы mergeUtil()
def merge(h1, h2):
    if (h1 == None):
        return h2

    if (h2 == None):
        return h1

    # Начнем со связанного списка, данные заголовка которого
наименьшие
    if (h1.data < h2.data):
        return mergeUtil(h1, h2)
    else:
        return mergeUtil(h2, h1)
```

39. Max Consecutive Ones III

Дан двоичный массив nums и целое число k, вернуть максимальное количество последовательных единиц в массиве, если вы можете поменять не более k нулей.

```
# Time O(n)
# Space O(1)
from typing import List

class Solution:
    def longestOnes(self, nums: List[int], k: int) -> int:
        if not nums or len(nums) == 0:
            return 0

        left, right = 0, 0
        if nums[right] == 0:
```

```
            k -= 1

        n = len(nums)
        max_length = 0

        for left in range(n):

            while right + 1 < n and ( (k > 0) or nums[right + 1] ==
1 ):
                if nums[right + 1] == 0:
                    k -= 1
                right += 1

            if k >= 0:
                max_length = max(max_length, right - left + 1)

            if nums[left] == 0:
                k += 1

        return max_length
```

40. Longest Palindromic Substring

Учитывая строку s, вернуть самый длинный палиндром

```
# Time O(n2)
# Space O(1)

def longestPalindrome(s):
    res = ""
    for i in range(len(s)):
        odd  = palindromeAt(s, i, i)
        even = palindromeAt(s, i, i+1)

        res = max(res, odd, even, key=len)
    return res

# начиная с l,r расширить наружу, чтобы найти самый большой палиндром
def palindromeAt(s, l, r):
    while l >= 0 and r < len(s) and s[l] == s[r]:
        l -= 1
        r += 1
    return s[l+1:r]
```

41. Jewels and Stones

Вам даны нити драгоценностей, представляющие типы камней, которые являются драгоценностями, и камни, представляющие камни, которые у вас есть. Каждый символ в камнях - это тип камня, который у вас есть. Вы

хотите знать, сколько из камней, которые у вас есть, также являются драгоценностями.

Буквы чувствительны к регистру, поэтому «a» считается отличным от «A» типом камня.

```
# Time O(n)
# Space O(1)

def count_jewels_in_stones2(J, S):
    s_set = set(J)
    count_jewels_in_collection = 0

    for item in S:
        if item in s_set:
            count_jewels_in_collection += 1
    return count_jewels_in_collection
```

42. Lowest Common Ancestor of a Binary Tree

Для заданного бинарного дерева найдите наименьшего общего предка (LCA) двух заданных узлов в дереве.

Согласно определению LCA в Википедии: «Наименьший общий предок определяется между двумя узлами p и q как наименьший узел в T, у которого есть потомки p и q (где мы позволяем узлу быть потомком самого себя). ”

```
# Time O(n)
# Space O(n)

class LowestCommonAncestorFinder:
    def lowestCommonAncestor(self, root, p, q):
        stack = [root]
        parent = {root: None}
        while stack:
            node = stack.pop()
            if node.left:
                parent[node.left] = node
                stack.append(node.left)
            if node.right:
                parent[node.right] = node
                stack.append(node.right)
        ancestors = set()
        while p:
            ancestors.add(p)
            p = parent[p]
        while q not in ancestors:
            q = parent[q]
        return q
```

43. Intersection Of Two Arrays ii

Даны два целочисленных массива nums1 и nums2, вернуть массив их пересечения. Каждый элемент в результате должен появляться столько раз, сколько он отображается в обоих массивах, и вы можете возвращать результат в любом порядке.

```
class Solution(object):
    def intersect(self, nums1, nums2):

        counts = {}
        res = []

        for num in nums1:
            counts[num] = counts.get(num, 0) + 1

        for num in nums2:
            if num in counts and counts[num] > 0:
                res.append(num)
                counts[num] -= 1

        return res

# Time O(mlogm + nlogn)
# Space O(1)

class Solution:
    def intersect(self, nums1, nums2):
        # подход: отсортировать два списка и использовать два
указателя для сравнения

        nums1.sort()
        nums2.sort()

        p1 = p2 = 0
        result = []

        while p1 < len(nums1) and p2 < len(nums2):
            num1 = nums1[p1]
            num2 = nums2[p2]

            if num1 < num2:
                p1 += 1
            elif num1 > num2:
                p2 += 1
            else:
                result.append(num1)
                p1 += 1
                p2 += 1
```

```
        return result
```

44. Missing number

Дан массив nums, содержащий n различных чисел в диапазоне [0, n], вернуть единственное число в диапазоне, отсутствующее в массиве.

```
class Solution:
    def missingNumber(self, nums: List[int]) -> int:
        n = len(nums)
        n_elements_sum = n * (n+1) // 2  # Формула суммы чисел от 1 до
n
        return n_elements_sum - sum(nums)
```

45. Evaluate Reverse Polish Notation

```
# Time O(n)
# Space O(n)

import math

class Solution:
    def evalRPN(self, tokens: List[str]) -> int:
        stack = []
        operators = {'+', '-', '*', '/'}

        if len(tokens) == 1:
            return int(tokens[0])
        for token in tokens:
            if token not in operators:
                stack.append(int(token))
            else:
                sec_operand = stack.pop()
                fir_operand = stack.pop()
                tmp = 0
                if token == '+':
                    tmp = fir_operand + sec_operand
                elif token == '-':
                    tmp = fir_operand - sec_operand
                elif token == '*':
                    tmp = fir_operand * sec_operand
                elif token == '/':
                    tmp = fir_operand / sec_operand
                    # tmp = math.trunc(tmp)
                stack.append(tmp)

        return stack.pop()
```

46. Median of two sorted arrays

Даны два отсортированных массива nums1 и nums2 размера m и n соответственно, вернуть медиану двух отсортированных массивов.

Общая сложность времени выполнения должна быть O(log (m+n)).

```
def find(nums1, s1, e1, nums2, s2, e2, k):
    if e1 - s1 < 0:
        return nums2[k + s2]
    if e2 - s2 < 0:
        return nums1[k + s1]
    if k < 1:
        return min(nums1[k + s1], nums2[k + s2])

    ia, ib = (s1 + e1) // 2 , (s2 + e2) // 2
    ma, mb = nums1[ia], nums2[ib]
    if (ia - s1) + (ib - s2) < k:
        if ma > mb:
            return find(nums1, s1, e1, nums2, ib + 1, e2, k - (ib -
s2) - 1)
        else:
            return find(nums1, ia + 1, e1, nums2, s2, e2, k - (ia -
s1) - 1)
    else:
        if ma > mb:
            return self.find(nums1, s1, ia - 1, nums2, s2, e2, k)
        else:
            return self.find(nums1, s1, e1, nums2, s2, ib - 1, k)

def findMedianSortedArrays(nums1, nums2):
    l = len(nums1) + len(nums2)
    if l % 2 == 1:
        return self.find(nums1, 0, len(nums1) - 1, nums2, 0,
len(nums2) - 1, l // 2)
    else:
        return (self.find(nums1, 0, len(nums1) - 1, nums2, 0,
len(nums2) - 1, l // 2)
              + self.find(nums1, 0, len(nums1) - 1, nums2, 0,
len(nums2) - 1, l // 2 - 1)) / 2.0
```

47. Simplify Path

- Input: path = "/home//foo/"
- Output: "/home/foo"

```
class Solution:
    def simplifyPath(self, path: str) -> str:
        path = path.split('/')
        res = ""
        res_stack = []
        for index, char in enumerate(path):
```

```
            if char == "..":
                if res_stack:
                    res_stack.pop()
            elif char == "." or char == "":
                pass
            else:
                res_stack.append('/' + char)
        if len(res_stack) == 0:
            return "/"
        return res.join(res_stack)
```

48. Is Subsequence

Имея две строки s и t, вернуть true, если s является подпоследовательностью t, или false в противном случае.

Подпоследовательность строки — это новая строка, образованная из исходной строки путем удаления некоторых (может быть ни одного) символов без нарушения взаимного расположения оставшихся символов. (т. е. «ace» является подпоследовательностью «abcde», а «aec» — нет).

```
# Time O(n)
# Space O(1)

class Solution:
    def isSubsequence(self, s: str, t: str) -> bool:
        if not s:
            return True
        i = 0

        for c in t:
            if c == s[i]:
                i += 1
            if i >= len(s):
                break

        if i == len(s):
            return True

        return False
```

49. Squares Of A Sorted Array

Дан целочисленный массив nums, отсортированный в неубывающем порядке, вернуть массив квадратов каждого числа, отсортированного в неубывающем порядке.

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

```
# Time O(n)

def sortedSquares(self, A):
    answer = [0] * len(A)
    l, r = 0, len(A) - 1

    while l <= r:
        left, right = abs(A[l]), abs(A[r])
        if left > right:
            answer[r - l] = left * left
            l += 1
        else:
            answer[r - l] = right * right
            r -= 1
    return answer
```

50. Remove Nth Node From End of List

```
# Time O(n)

class Solution:
    def removeNthFromEnd(self, head: Optional[ListNode], n: int) ->
Optional[ListNode]:
        fast, slow = head, head
        for _ in range(n):
            fast = fast.next

        if not fast:
            return head.next

        while fast.next:
            fast, slow = fast.next, slow.next
        slow.next = slow.next.next
        return head
```

51. Perfect Squares

Given an integer n, return the least number of perfect square numbers that sum to n.

- Input: n = 12
- Output: 3
- Explanation: 12 = 4 + 4 + 4.

Основная идея этого решения — поиск BSF кратчайшего пути, в качестве примера возьмем 12, как показано ниже, кратчайший путь — 12-8-4-0:

```
# Time O(n)
# Space O(1)

def numSquares(self, n):
    if n < 2:
```

```
        return n

    lst = []
    i = 1

    while i**2 <= n:
        lst.append(i**2)
        i += 1

    cnt = 0
    to_check = {n}

    while to_check:
        cnt += 1
        temp = set()
        for x in to_check:
            for y in lst:
                if x == y:
                    return cnt
                if x < y:
                    break
                temp.add(x-y)

        to_check = temp

    return cnt
```

52. Max Stack

Спроектируйте стек, который поддерживает push, pop, top, peekMax и popMax.

- push(x) — Поместить элемент x в стек.
- pop() — удалить элемент с вершины стека и вернуть его.
- top() — Получить элемент сверху.
- peekMax() — Получить максимальный элемент в стеке.
- popMax() — получает максимальный элемент в стеке и удаляет его. Если вы найдете более одного максимального элемента, удалите только самый верхний.

```
class MaxStack(object):

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

    def push(self, x):
        self.stack.append(x)
        if len(self.max_stack) == 0:
            self.max_stack.append(x)
            return
```

```
        if self.max_stack[-1] > x:
            self.max_stack.append(self.max_stack[-1])
        else:
            self.max_stack.append(x)

    def pop(self):
        if len(self.stack) != 0:
            self.max_stack.pop(-1)
            return self.stack.pop(-1)

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

    def peekMax(self):
        if len(self.max_stack) != 0:
            return self.max_stack[-1]

    def popMax(self):
        val = self.peekMax()
        buff = []

        while self.top() != val:
            buff.append(self.pop())
        self.pop()

        while len(buff) != 0:
            self.push(buff.pop(-1))
        return val
```

53. Lowest common ancestor of a binary tree iii

Учитывая два узла бинарного дерева nodeA и nodeB, вернуть их наименьший общий предок (LCA). Каждый узел имеет следующие атрибуты: int val, Node left, Node right, int parent.

Алгоритм

Этот метод немного сложен, и большинство людей не смогут придумать его за короткий промежуток времени.

Идея такова: мы будем использовать 2 указателя (pointerA, pointerB), которые идут от nodeA и nodeB вверх соответственно.

Предположим, что узел A расположен на более мелком уровне, чем узел B, т. е. глубина (узел A) < глубины (узел B), указатель A достигнет вершины быстрее, чем указатель B.

Предположим, что разница в глубине между узлом A и узлом B равна diff, к тому времени, когда указатель A достигнет вершины, указатель B будет

отставать от него на разные уровни. Теперь, если указатель A сбрасывает свой путь и продолжает движение вверх от узла B вместо узла A, ему потребуются различные шаги, чтобы достичь уровня узла A, к тому времени указатель B уже догнал и будет на том же уровне указателя A (указатель B перезапустится с узла A после достигнув вершины).

Теперь единственное, что нужно сделать, это сравнить указатель A и указатель B на пути вверх. Если pointerA и pointerB указывают на один и тот же узел, мы нашли самого младшего общего предка.

```
# Time O(n)
# Space O(1)

def lowestCommonAncestor(self, a, b):
    pointerA, pointerB = a, b

    while pointerA != pointerB:
        pointerA = pointerA.parent if pointerA else b
        pointerB = pointerB.parent if pointerA else a

    return pointerA
```

Алгоритм

Один из способов решить эту проблему — пройти от одного из узлов до самого верха и сохранить всех предков в хэш-карте. После этого мы перейдем от другого узла к вершине дерева и вернем узел, если он уже есть в хэш-карте. Если да, то мы нашли низшего общего предка.

```
# Time O(n)
# Space O(n)

def lowestCommonAncestor(self, a: 'Node', b: 'Node') -> 'Node':
    ancestors = set()

    while a is not None:
        ancestors.add(a)
        a = a.parent

    while b is not None:
        if b in ancestors:
            return b
        b = b.parent
```

54. Longest substring with at most two distinct characters

```
# Time:  O(n)
# Space: O(1)

class Solution():
    def lengthOfLongestSubstringTwoDistinct(self, s):
```

```
        longest, start, distinct_count, visited = 0, 0, 0, [0 for _ in
range(256)]

        for i, char in enumerate(s):
            if visited[ord(char)] == 0:
                distinct_count += 1

            visited[ord(char)] += 1

            while distinct_count > 2:
                visited[ord(s[start])] -= 1
                if visited[ord(s[start])] == 0:
                    distinct_count -= 1
                start += 1

            longest = max(longest, i - start + 1)
        return longest
```

55. Maximal Rectangle

Дана бинарная матрица строк x столбцов, заполненная нулями и единицами, найти самый большой прямоугольник, содержащий только единицы, и вернуть его площадь.

Наконец, есть очень умный способ со сложностью O(nm). Действительно, **для каждой строки мы можем вычислить высоту нашей линии горизонта, учитывая предыдущую строку за O(m) и сделать это n раз**. Сложность по памяти — O(m), потому что на самом деле нам нужно хранить только одну строку за раз.

```
# Time O(nm)
# Space O(m)

class Solution:
    def maximalRectangle(self, matrix):
        def hist(heights):
            stack, ans = [], 0
            for i, h in enumerate(heights + [0]):
                while stack and heights[stack[-1]] >= h:
                    H = heights[stack.pop()]
                    W = i if not stack else i-stack[-1]-1
                    ans = max(ans, H*W)
                stack.append(i)
            return ans

        if not matrix or not matrix[0]:
            return 0
        m, n, ans = len(matrix[0]), len(matrix), 0
        row = [0] * m
        for i in range(n):
```

```
            for j in range(m):
                row[j] = 0 if matrix[i][j] == "0" else row[j] + 1
            ans = max(ans, hist(row))

        return ans
```

56. Search in Rotated Sorted Array

- Input: nums = [4,5,6,7,0,1,2], target = 0
- Output: 4

```
# Time O(logn)
# Space O(1)

class Solution:
    def search(self, nums, target):
        low = 0
        high = len(nums)

        while low < high:
            mid = low + (high-low) // 2

            if nums[mid] == target:
                return mid
            if nums[low] <= nums[mid]:
                if target >= nums[low] and target < nums[mid]:
                    high = mid
                else:
                    low = mid + 1
            else:
                if target <= nums[high-1] and target > nums[mid]:
                    low = mid + 1
                else:
                    high = mid
        return -1
```

57. Two Sum II - Input Array Is Sorted

- Input: numbers = [2,7,11,15], target = 9
- Output: [1,2]
- Explanation: The sum of 2 and 7 is 9. Therefore, index1 = 1, index2 = 2. We return [1, 2].

```
# Time O(n)
# Space O(1)

class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[int]:

        i = 0
        j = len(numbers) - 1
```

```
        while i < j:
            if numbers[i] + numbers[j] < target:
                i += 1
            elif numbers[i] + numbers[j] > target:
                j -= 1
            else:
                return [i + 1, j + 1]

        return []
```

58. Remove Duplicates from Sorted Array - Space O(1)

- Input: nums = [0,0,1,1,1,2,2,3,3,4]
- Output: 5, nums = [0,1,2,3,4,_,_,_,_,_]

```
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        if len(nums) == 0:
            return 0

        length = 1
        previous = nums[0]
        index = 1

        for i in range(1, len(nums)):
            if nums[i] != previous:
                length += 1
                previous = nums[i]
                nums[index] = nums[i]
                index+=1
        return length
```

59. Find minium in rotated sorted array

[3,4,5,1,2] -> 1

```
from typing import List

# Time O(logn)
# Space O(1)

class Solution:
    def findMin(self, nums: List[int]) -> int:
        # Если в списке только один элемент, вернуть этот элемент.
        if len(nums) == 1:
            return nums[0]


        left = 0
        right = len(nums) - 1

        # если последний элемент больше первого, то вращения нет.
```

```
        # например 1 < 2 < 3 < 4 < 5 < 7. Уже отсортированный массив.
        # Следовательно, наименьший элемент является первым элементом.
A[0]
        if nums[right] > nums[0]:
            return nums[0]

        while right >= left:

            mid = left + (right - left) // 2

            # если средний элемент больше, чем его следующий элемент,
            # то элемент mid+1 является наименьшим
            # Эта точка будет точкой изменения.
            # От большего значения к меньшему.
            if nums[mid] > nums[mid + 1]:
                return nums[mid + 1]

            # если средний элемент меньше предыдущего,
            # то средний элемент наименьший
            if nums[mid - 1] > nums[mid]:
                return nums[mid]

            # если значение среднего элемента больше,
            # чем 0-й элемент, это означает
            # наименьшее значение по-прежнему где-то справа,
            # так как мы по-прежнему имеем дело с элементами больше,
чем nums[0]
            if nums[mid] > nums[0]:
                left = mid + 1

            # если nums[0] больше среднего значения,
            # это означает, что наименьшее значение находится где-то
слева
            else:
                right = mid - 1
```

60. Range sum of BST

вернуть сумму значений всех узлов со значением в инклюзивном диапазоне [low, high].

Мы проходим по дереву, используя поиск в глубину. Если node.val выходит за пределы диапазона [low, high] (например, node.val < low), то мы знаем, что только правая ветвь может иметь узлы со значением внутри [low, high].

Мы демонстрируем две реализации — одну с использованием рекурсивного алгоритма, а другую — с итеративным.

```
from typing import Optional
```

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def rangeSumBST(self, root: Optional[TreeNode], low: int, high:
int) -> int:
        ans = 0
        stack = [root]
        while stack:
            node = stack.pop()
            if node:
                if low <= node.val <= high:
                    ans += node.val
                if low < node.val:
                    stack.append(node.left)
                if node.val < high:
                    stack.append(node.right)
        return ans
```

61. Partition Labels

Вам дана строка s. Мы хотим разделить строку на как можно больше частей, чтобы каждая буква встречалась не более чем в одной части.

Обратите внимание, что разбиение сделано таким образом, что после объединения всех частей по порядку результирующая строка должна быть s.

Возвращает список целых чисел, представляющих размер этих частей.

```
class Solution:
    # Time: O(n), Space: O(1)
    def partitionLabels(self, s: str) -> List[int]:
        helper_dict = {}
        for i, char in enumerate(s):
            helper_dict[char] = i

        left, right = 0, 0
        res = []

        for i, char in enumerate(s):
            right = max(right, helper_dict[char])
            if i == right:
                res.append(right-left+1)
                left = right + 1

        return res
```

62. Product of array except self

Учитывая целочисленный массив nums, вернуть такой массив, что answer[i] равен произведению всех элементов nums, кроме nums[i].

Произведение любого префикса или суффикса чисел гарантированно соответствует 32-битному целому числу.

Вы должны написать алгоритм, который выполняется за время **O(n)** и не использует операцию деления.

```
class Solution:
    def productExceptSelf(self, nums: List[int]) -> List[int]:
        ans = [1 for _ in nums]

        left = 1
        right = 1

        for i in range(len(nums)):
            ans[i] *= left
            ans[-1-i] *= right
            left *= nums[i]
            right *= nums[-1-i]

        return ans
```

63. Binary Tree Maximum Path Sum

Путь в бинарном дереве — это последовательность узлов, в которой каждая пара соседних узлов в последовательности имеет соединяющее их ребро. Узел может появиться в последовательности не более одного раза. Обратите внимание, что путь не обязательно должен проходить через корень.

Сумма путей пути — это сумма значений узлов пути.

Учитывая корень двоичного дерева, **вернуть максимальную сумму пути любого непустого пути**.

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

res = [3, 9, 20, 15, 7]

```
class Solution:
    def max_path_sum(self, root: Optional[TreeNode]) -> int:
        max_path = -float('inf')
```

```
        # обход по порядку поддерева с корнем в `node`
        def gain_from_subtree(node: Optional[TreeNode]) -> int:
            nonlocal max_path

            if not node:
                return 0

            # добавляем усиление из левого поддерева. Обратите
внимание, что если
            # коэффициент усиления отрицателен, его можно игнорировать
или считать равным 0.
            # Вот почему мы используем здесь `max`.
            gain_from_left = max(gain_from_subtree(node.left), 0)

            # добавить сумму усиления/пути из правого поддерева. 0,
если отрицательный
            gain_from_right = max(gain_from_subtree(node.right), 0)

            # если левое или правое усиление отрицательно, они
учитываются
            # как 0, так что этот оператор заботится обо всех четырех
сценариях
            max_path = max(max_path, gain_from_left + gain_from_right
+ node.val)

            # вернуть максимальную сумму для пути, начинающегося в
корне поддерева
            return max(
                gain_from_left + node.val,
                gain_from_right + node.val
            )

        gain_from_subtree(root)
        return max_path
```

64. Continuous Subarray Sum

Дан массив и число. Определить, содержит ли массив хороший подмассив.

Хороший подмассив — это подмассив, в котором:

- длина не менее двух, а сумма элементов подмассива кратна k.

Обратите внимание, что:

- Подмассив — это непрерывная часть массива.
- Целое число x кратно k, если существует целое число n такое, что x = n * k. 0 всегда кратно k.

```
# Time O(n)
# Space O(min{n, k})
```

```python
class Solution:
    def checkSubarraySum(self, nums: List[int], k: int) -> bool:
        # инициализируем словарь индексом 0 для суммы 0
        hash_map = {0: 0}
        s = 0
        for i in range(len(nums)):
            s += nums[i]
            # если остаток s % k встречается впервые
            if s % k not in hash_map:
                hash_map[s % k] = i + 1

            # если размер подмассива не меньше двух
            elif hash_map[s % k] < i:
                return True
        return False
```

65. Reverse Words in a String III

Для строки s изменить порядок символов в каждом слове в предложении, сохраняя при этом пробелы и первоначальный порядок слов.

- Input: s = "God Ding"
- Output: "doG gniD"

```python
def reverseWords_manual(s):  # O(n) both
    res = ''
    l, r = 0, 0
    while r < len(s):
        if s[r] != ' ':
            r += 1
        elif s[r] == ' ':
            res += s[l:r + 1][::-1]
            r += 1
            l = r
    res += ' '
    res += s[l:r + 2][::-1]
    return res[1:]

def reverseWords(self, s):
    return ' '.join(x[::-1] for x in s.split())
```

66. Add Strings

Сложить строки как числа

```python
# Time O(m+n)
class Solution:
    def addStrings(self, num1: str, num2: str) -> str:
        ans = []
        i1, i2 = len(num1) - 1, len(num2) - 1
        carry = 0
```

```
        while i1 >= 0 or i2 >= 0 or carry > 0:
            if i1 >= 0:
                carry += ord(num1[i1]) - ord('0')
                i1 -= 1
            if i2 >= 0:
                carry += ord(num2[i2]) - ord('0')
                i2 -= 1
            ans.append(chr(carry % 10 + ord('0')))
            carry //= 10
        return "".join(ans)[::-1]
```

67. Palindrome Linked List

Является ли список палиндромом?

- Input: head = [1,2,2,1]
- Output: true

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class Solution:
    def isPalindrome(self, head: Optional[ListNode]) -> bool:
        slow, fast, prev = head, head, None

        while fast and fast.next:
            slow, fast = slow.next, fast.next.next
        prev, slow, prev.next = slow, slow.next, None

        while slow:
            temp = slow.next
            slow.next, prev, slow = prev, slow, temp

        fast, slow = head, prev

        while slow:
            if fast.val != slow.val: return False
            fast, slow = fast.next, slow.next
        return True
```

68. First Unique Character in a String

Для заданной строки s **найти в ней первый неповторяющийся символ** и вернуть его индекс. Если он не существует, верните -1.

```
# Time O(n)
# Space O(1)

class Solution:
```

```python
    def firstUniqChar(self, s: str) -> int:
        cnt = {}
        for c in s:
            if c not in cnt:
                cnt[c] = 1
            else:
                cnt[c] += 1

        for i in range(len(s)):
            if cnt[s[i]] == 1:
                return i

        return -1

class Solution(object):
    def firstUniqChar(self, s):
        for i in range(len(s)):
            if s[i] not in s[:i]+s[i+1:]:
                return i
        return -1
```