Гайд по найму: https://yandex.ru/recruitment-guide/backend-developers

Оценка сложности алгоритмов: https://habr.com/ru/post/188010/

```
from typing import Optional, List
from collections import deque
```

Связанные списки

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

1. Вам дан массив из k списков связанных списков, каждый связанный список отсортирован в порядке возрастания. Объедините все связанные списки в один отсортированный связанный список и верните его.

```
def SortedMerge(a, b):
        result = None
        if a == None:
            return b
        elif b == None:
            return a
        if a.val <= b.val:
            result = a
            result.next = SortedMerge(a.next, b)
        else:
            result = b
            result.next = SortedMerge(a, b.next)
        return result

def mergeKLists(arr: List[Optional[ListNode]]) -> Optional[ListNode]:
        if len(arr) == 0:
            return None
        last = len(arr) - 1
        while last != 0:
            i = 0
            j = last
            while i < j:
                arr[i] = SortedMerge(arr[i], arr[j])
                i += 1
                j -= 1
                if i >= j:
                    last = j
        return arr[0]
```

1. Дан head, заголовок связанного списка, определите, есть ли в связанном списке цикл. В связанном списке есть цикл, если в списке

есть какой-то узел, к которому можно снова добраться, непрерывно следуя за указателем next. Внутри pos используется для обозначения индекса узла, к которому подключен следующий указатель tail. Обратите внимание, что pos не передается в качестве параметра. Возвращает true, если в связанном списке есть цикл. В противном случае вернуть false.

```
# Time: O(n), Data: O(1)
def hasCycle(head: Optional[ListNode]) -> bool:
        temp = ""
        while head != None:
            if head.next == None:
                return False
            if head.next == temp:
                return True
            nxt = head.next
            head.next = temp
            head = nxt
        return False
```

1. Вам даны два непустых связанных списка, представляющих два неотрицательных целых числа. Цифры хранятся в обратном порядке, и каждый из их узлов содержит одну цифру. Добавьте два числа и верните сумму в виде связанного списка. Вы можете предположить, что эти два числа не содержат начальных нулей, кроме самого числа 0.

```
def addTwoNumbers(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
```

1. Дан head односвязного списка, переверните список и верните перевернутый список.

```
def reverseList(head: Optional[ListNode]) -> Optional[ListNode]:
        if head == None:
            return None
        stack, temp = [], head
        while temp:
            stack.append(temp)
            temp = temp.next
        head = temp = stack.pop()
        while len(stack) > 0:
            temp.next = stack.pop()
            temp = temp.next
        temp.next = None
        return head
```

Бинарный поиск

1. Дан массив целых чисел nums, отсортированный в порядке возрастания, и целочисленная цель, напишите функцию для поиска цели в nums. Если цель существует, верните ее индекс. В противном случае вернуть -1. Вы должны написать алгоритм со сложностью выполнения O(log n).

```
# Time: O(n), Space: O(1)
def search(nums: List[int], target: int) -> int:
        mid = 0
        start = 0
        end = len(nums) - 1
        while start <= end:
            mid = (start + end) // 2
            if target == nums[mid]:
                return mid
            if target < nums[mid]:
                end = mid - 1
            else:
                start = mid + 1
        return -1
```

1. Мы играем в игру «Угадай». Игра выглядит следующим образом: Я выбираю число от 1 до n. Вы должны угадать, какое число я выбрал. Каждый раз, когда вы угадаете неправильно, я скажу вам, выше или ниже выбранное вами число. Вы вызываете предопределенный API int guess(int num), который возвращает три возможных результата:

-1: Ваше предположение больше, чем число, которое я выбрал (т. е. число > выбрать)

1: Ваше предположение меньше числа, которое я выбрал (т. е. число < выбрано)

0: ваше предположение равно числу, которое я выбрал (т. е. num == pick)

```
def guessNumber(n: int) -> int:
        left, right = 1, n
        while left <= right:
            mid = left + (right - left) // 2
            if guess(mid) <= 0:
                right = mid - 1
            else:
                left = mid + 1
        return left
```

1. Напишите эффективный алгоритм, который ищет целевое значение в целочисленной матрице размера m x n. Эта матрица обладает следующими свойствами: 1. Целые числа в каждой строке сортируются слева направо. 2. Первое целое число каждой строки больше, чем последнее целое число предыдущей строки.

```
def searchMatrix(matrix: List[List[int]], target: int) -> bool:
            if not matrix:
                return False

            row, col = len(matrix), len(matrix[0])
            if row == 0 or col == 0:
                return False

            r, c = 0, col - 1
            for i in range(row):
                if target <= matrix[i][c]:
                    l, r = 0, col-1
                    while l <= r:
                        mid = (l + r) // 2
                        if matrix[i][mid] == target:
                            return True
                        if target < matrix[i][mid]:
                            r = mid-1
                        else:
                            l = mid+1
                    return False
                else:
                    continue
            return False
```

1. Существует целочисленный массив nums, отсортированный в порядке возрастания (с различными значениями). Перед тем, как быть переданным вашей функции, nums, возможно, вращается с неизвестным опорным индексом k (1 <= k < nums.length), так что результирующий массив равен [nums[k], nums[k+1],... , nums[n-1], nums[0], nums[1], ..., nums[k-1]] (с индексом 0). Например, [0,1,2,4,5,6,7] может быть повернут с поворотным индексом 3 и станет [4,5,6,7,0,1,2].

Учитывая массив nums после возможного поворота и целочисленную цель, вернуть индекс цели, если он находится в nums, или -1, если он не в nums. Вы должны написать алгоритм со сложностью выполнения O(log n).

```
# Time: O(logn), Space: O(1)
def search(nums: List[int], target: int) -> int:
        if nums is None or len(nums) == 0:
            return -1

        left, right = 0, len(nums) - 1

        while left < right:
            middle = left + (right - left) // 2
            if nums[middle] > nums[right]:
                left = middle + 1
            else:
                right = middle

        pivot = left
        left, right = 0, len(nums) - 1
        if nums[pivot] <= target <= nums[right]:
            left = pivot
        else:
            right = pivot
        while left <= right:
            middle = left + (right - left) // 2
            if nums[middle] == target:
                return middle
            elif nums[middle] < target:
                left = middle + 1
            else:
                right = middle - 1
        return -1
```

1. Предположим, что массив длины n, отсортированный в порядке возрастания, вращается от 1 до n раз. Например, массив nums = [0,1,2,4,5,6,7] может стать следующим:

[4,5,6,7,0,1,2], если он был повернут 4 раза. [0,1,2,4,5,6,7], если он был повернут 7 раз. Обратите внимание, что поворот массива [a[0], a[1], a[2], ..., a[n-1]] 1 раз приводит к массиву [a[n-1], a[0] , a[1], a[2], ..., a[n-2]].

Учитывая число уникальных элементов отсортированного повернутого массива, вернуть минимальный элемент этого массива. Вы должны написать алгоритм, который работает за время O(log n).

```
def findMin(nums: List[int]) -> int:
        low = 0
        high = len(nums) - 1
        while (low < high):
```

```
            mid = low + (high - low) // 2

            if (nums[mid] == nums[high]):
                high -= 1
            elif (nums[mid] > nums[high]):
                low = mid + 1
            else:
                high = mid
        return nums[high]
```

1. Существует целочисленный массив nums, отсортированный в порядке неубывания (не обязательно с различными значениями). Перед передачей в вашу функцию nums вращается с неизвестным опорным индексом k (0 <= k < nums.length), так что результирующий массив равен [nums[k], nums[k+1], ..., nums [n-1], nums[0], nums[1], ..., nums[k-1]] (с индексом 0). Например, [0,1,2,4,4,4,5,6,6,7] можно повернуть с опорным индексом 5 и превратить в [4,5,6,6,7,0,1,2, 4,4]. Учитывая массив nums после поворота и целочисленную цель, вернуть true, если цель находится в nums, или false, если она не в nums. Вы должны максимально сократить общие шаги операции.

```
# Time: O(logn), Space: O(1)
def search(nums: List[int], target: int) -> bool:
        pivot = 0
        for ar in range(len(nums)-1):
            if nums[ar] > nums[ar+1]:
                pivot = ar + 1
                break
        nums = nums[pivot:] + nums[0:pivot]
        left = 0
        right = len(nums) - 1
        while left <= right:
            mid = (left+right) // 2
            if nums[mid] == target:
                return True
            elif nums[mid] > target:
                right = mid - 1
            else:
                left = mid + 1
        return False
```

Хэш таблицы

1. Учитывая непустой массив целых чисел nums, каждый элемент появляется дважды, кроме одного. Найди ту единственную. Вы должны реализовать решение с линейной сложностью времени выполнения и использовать только постоянное дополнительное пространство.

```
def singleNumber(nums: List[int]) -> int:
        res = nums[0]
        n = len(nums)
        for i in range(1, n):
            res = res ^ nums[i]
        return res
```

1. Учитывая массив целых чисел nums и целочисленную цель, вернуть индексы двух чисел так, чтобы они складывались в цель. Вы можете предположить, что каждый вход будет иметь ровно одно решение, и вы не можете использовать один и тот же элемент дважды. Вы можете вернуть ответ в любом порядке.

```
def twoSumHashing(num_arr, pair_sum):
    sums = []
    hashTable = {}

    for i in range(len(num_arr)):
        complement = pair_sum - num_arr[i]
        if complement in hashTable:
            return num_arr[i], complement
        hashTable[num_arr[i]] = num_arr[i]
```

1. Учитывая массив nums из n целых чисел, вернуть массив всех уникальных четверок [nums[a], nums[b], nums[c], nums[d]], таких что:

0 <= a, b, c, d < n

a, b, c и d различны

nums[a] + nums[b] + nums[c] + nums[d] == target

Вы можете вернуть ответ в любом порядке.

```
def fourSum(nums: List[int], target: int) -> List[List[int]]:
        def findNsum(nums, target, n, result, results):
            if len(nums) < n or n < 2 or target < nums[0]*n or target
> nums[-1]*n:
                return
            if n == 2:
                l, r = 0, len(nums) - 1
                while l < r:
                    s = nums[l] + nums[r]
                    if s == target:
                        results.append(result + [nums[l], nums[r]])
                        l += 1
                        while l < r and nums[l] == nums[l-1]:
                            l += 1
                    elif s < target:
                        l += 1
                    else:
                        r -= 1
```

```
            else:
                for i in range(len(nums) - n + 1):
                    if i == 0 or (i > 0 and nums[i-1] != nums[i]):
                        findNsum(nums[i+1:], target-nums[i], n-1,
result+[nums[i]], results)

        results = []
        findNsum(sorted(nums), target, 4, [], results)
        return results
```

1. Учитывая массив строк strs, сгруппируйте анаграммы вместе. Вы можете вернуть ответ в любом порядке. Анаграмма — это слово или фраза, образованная путем перестановки букв другого слова или фразы, обычно с использованием всех исходных букв ровно один раз.

```
def groupAnagrams(li: List[str]) -> List[List[str]]:
        dictionary = {}
        for word in li:
            sortedWord = ''.join(sorted(word))

            if sortedWord not in dictionary:
                dictionary[sortedWord] = [word]

            else:
                dictionary[sortedWord] += [word]
        return [dictionary[i] for i in dictionary]
```

1. Имея две строки s и t, вернуть true, если t является анаграммой s, и false в противном случае. Анаграмма — это слово или фраза, образованная путем перестановки букв другого слова или фразы, обычно с использованием всех исходных букв ровно один раз.

```
def isAnagram(str1: str, str2: str) -> bool:
        NO_OF_CHARS = 256

        if len(str1) != len(str2):
            return False

        count = [0 for i in range(NO_OF_CHARS)]
        i = 0

        for i in range(len(str1)):
            count[ord(str1[i]) - ord('a')] += 1;
            count[ord(str2[i]) - ord('a')] -= 1;

        for i in range(NO_OF_CHARS):
            if (count[i] != 0):
                return False

        return True
```

1. Имея две строки s и p, вернуть массив всех начальных индексов анаграмм p в s. Вы можете вернуть ответ в любом порядке. Анаграмма — это слово или фраза, образованная путем перестановки букв другого слова или фразы, обычно с использованием всех исходных букв ровно один раз.

```
def findAnagrams(self, s: str, p: str) -> List[int]:
        cnt = Counter(p)
        res = []
        for i in range(len(s)):
            if s[i] in cnt:
                cnt[s[i]] -= 1
            if i - len(p) >= 0 and s[i-len(p)] in cnt:
                cnt[s[i-len(p)]] += 1
            if all([i == 0 for i in cnt.values()]):
                res.append(i-len(p)+1)
        return res
```

Очередь/стек

1. Учитывая строку s, содержащую только символы '(', ')', '{', '}', '[' и ']', определите, допустима ли входная строка. Входная строка действительна, если:

Открытые скобки должны быть закрыты однотипными скобками.

Открытые скобки должны быть закрыты в правильном порядке.

```
# Time: O(n) / Space: O(n)
def isValid(s: str) -> bool:
        stack = []
        for char in s:
            if char in ["(", "{", "["]:
                stack.append(char)
            else:
                if not stack:
                    return False
                current_char = stack.pop()
                if current_char == '(':
                    if char != ")":
                        return False
                if current_char == '{':
                    if char != "}":
                        return False
                if current_char == '[':
                    if char != "]":
                        return False

        if stack:
```

```
            return False
        return True
```

DFS/BFS

1. Учитывая двумерную двоичную сетку m x n, которая представляет собой карту «1» (земля) и «0» (вода), верните количество островов. Остров окружен водой и образован путем соединения соседних земель по горизонтали или вертикали. Вы можете предположить, что все четыре края сетки окружены водой.

```
def numIslands(grid: List[List[str]]) -> int:
        def dfs(x, y):
            if x < 0 or x >= len(grid) or y < 0 or y >= len(grid[0])
or grid[x][y] != '1':
                return
            grid[x][y] = '0'
            for i, j in [(x - 1, y), (x, y - 1), (x + 1, y), (x, y +
1)]:
                dfs(i, j)

        if not grid or not grid[0]:
            return 0
        res = 0
        for i in range(len(grid)):
            for j in range(len(grid[0])):
                if grid[i][j] == '1':
                    dfs(i, j)
                    res += 1
        return res
```

1. Учитывая строку s, которая содержит круглые скобки и буквы, удалите минимальное количество недопустимых круглых скобок, чтобы сделать входную строку допустимой. Верните все возможные результаты. Вы можете вернуть ответ в любом порядке.

```
# Time: O(n), Space: O(n)
class Solution:
    def removeInvalidParentheses(self, s: str) -> List[str]:
        results = []

        left, right = self.left_right_count(s)
        self.dfs_helper(s, left, right, 0, results)

        return results


    def left_right_count(self, s):
        """
        Находит разницу кол-ва левых и правых скобок
        """
```

```
        left = 0
        right = 0

        for c in s:
            if c == '(':
                left += 1
            elif c == ')':
                if left > 0:
                    left -= 1
                else:
                    right += 1

        return left, right


    def is_valid(self, s):
        left, right = self.left_right_count(s)
        return left == 0 and right == 0

    def dfs_helper(self, s, left, right, start, results):
        if left == 0 and right == 0:
            if self.is_valid(s):
                results.append(s)
                return

        for i in range(start, len(s)):
            if i > start and s[i] == s[i - 1]:
                continue

            if s[i] == '(':
                self.dfs_helper(s[:i] + s[i + 1:], left - 1, right, i,
results)
            elif s[i] == ')':
                self.dfs_helper(s[:i] + s[i + 1:], left, right - 1, i,
results)
```

Сортировка

1. Учитывая массив интервалов, где intervals[i] = [start_i, end_i], объединить все перекрывающиеся интервалы и вернуть массив непересекающихся интервалов, которые охватывают все интервалы во входных данных.

```
# Time: O(nlogn) / Space: O(1)
def merge(arr: List[List[int]]) -> List[List[int]]:
        arr.sort(key = lambda x: x[0])
        m = []
        s = -10000
        max = -100000
```

```
        for i in range(len(arr)):
            a = arr[i]
            if a[0] > max:
                if i != 0:
                    m.append([s,max])
                max = a[1]
                s = a[0]
            else:
                if a[1] >= max:
                    max = a[1]

        if max != -100000 and [s, max] not in m:
            m.append([s, max])

        return m
```

Хэш/куча

1. Учитывая массив строк words и целое число k, вернуть k наиболее часто встречающихся строк. Верните ответ, отсортированный по частоте от самой высокой до самой низкой. Отсортируйте слова с одинаковой частотностью в лексикографическом порядке.

```
def topKFrequent(words: List[str], k: int) -> List[str]:
        results = []

        count = {}
        for word in words:
            count[word] = count.get(word, 0) + 1

        heap = []
        for key, value in count.items():
            heapq.heappush(heap, (-value, key))

        for i in range(k):
            results.append(heapq.heappop(heap)[1])

        return results
```

1. Учитывая целочисленный массив nums и целое число k, вернуть k наиболее часто встречающихся элементов. Вы можете вернуть ответ в любом порядке.

```
def topKFrequent(nums: List[int], k: int) -> List[int]:
        mp = dict()
        n = len(nums)

        for i in range(0, n):
            if nums[i] not in mp:
                mp[nums[i]] = 0
            else:
```

```
            mp[nums[i]] += 1

        heap = [(value, key) for key, value in mp.items()]

        largest = heapq.nlargest(k, heap)

        res = []

        for i in range(k):
            print(res.append(largest[i][1]))
        return res
```

Два указателя

1. Вам дан целочисленный массив высоты длины n. Нарисовано n вертикальных линий, две конечные точки i-й линии равны (i, 0) и (i, height[i]). Найдите две линии, которые вместе с осью абсцисс образуют контейнер, содержащий наибольшее количество воды. Возвращает максимальное количество воды, которое может храниться в контейнере. Обратите внимание, что вы не можете наклонять контейнер.

```
def maxArea(height: List[int]) -> int:
        maxarea = 0
        left = 0
        right = len(height) - 1

        while left < right:
            width = right - left
            maxarea = max(maxarea, min(height[left], height[right]) *
width)
            if height[left] <= height[right]:
                left += 1
            else:
                right -= 1

        return maxarea
```

1. 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.

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

```
        left, right = 0, 0
        res = []

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

        return res
```

Скользящее окно

1. Медиана — это среднее значение в упорядоченном списке целых чисел. Если размер списка четный, среднего значения нет. Таким образом, медиана — это среднее значение двух средних значений.

Например, если arr = [2,3,4], медиана равна 3.

Например, если arr = [1,2,3,4], медиана равна (2 + 3)/2 = 2,5.

Вам дан целочисленный массив nums и целое число k. Существует скользящее окно размера k, которое движется от самого левого края массива до самого правого. Вы можете видеть только k чисел в окне. Каждый раз скользящее окно перемещается вправо на одну позицию.

Возврат медианного массива для каждого окна в исходном массиве. Будут приняты ответы в пределах 10-5 от фактического значения.

```
from bisect import bisect_left

def medianSlidingWindow(nums: List[int], k: int) -> List[float]:
    def get(tmp):
        l = len(tmp)
        if l & 1 == 1:
            return tmp[l // 2]
        else:
            return tmp[l // 2 - 1] * 0.5 + tmp[l // 2] * 0.5

        n = len(nums)
        res = []
        tmp = sorted(nums[:k])

        res.append(self.get(tmp) * 1.0)
        for i in range(n - k):
            index = bisect_left(tmp, nums[i])
            del tmp[index]
            bisect.insort(tmp, nums[i+k])
```

```
            res.append(self.get(tmp) * 1.0)
        return res
```

1. Вам дан массив целых чисел nums, есть скользящее окно размера k, которое движется от самого левого края массива до самого правого. Вы можете видеть только k чисел в окне. Каждый раз скользящее окно перемещается вправо на одну позицию. Вернуть максимальное скользящее окно.

```
# Time: O(n) / Space: O(k)
def maxSlidingWindow(arr: List[int], k: int) -> List[int]:
        Qi = deque()
        n = len(arr)
        for i in range(k):
            while Qi and arr[i] >= arr[Qi[-1]]:
                Qi.pop()
            Qi.append(i);

        result = []

        for i in range(k, n):
            result.append(arr[Qi[0]])
            while Qi and Qi[0] <= i-k:
                Qi.popleft()
            while Qi and arr[i] >= arr[Qi[-1]]:
                Qi.pop()
            Qi.append(i)

        result.append(arr[Qi[0]])

        return result
```

1. Вам дана строка s и целое число k. Вы можете выбрать любой символ строки и заменить его на любой другой заглавный английский символ. Вы можете выполнить эту операцию не более k раз. Верните длину самой длинной подстроки, содержащей ту же букву, которую вы можете получить после выполнения вышеуказанных операций.

```
def characterReplacement(s: str, k: int) -> int:
        start = end = res = maxcur = 0
        count = [0] * 26
        while end < len(s):
            count[ord(s[end])-65] += 1
            maxcur = max(maxcur, max(count))
            while end - start + 1 - maxcur > k:
                  count[ord(s[start])-65] -= 1
                  start += 1
            res = max(res, end-start+1)
            end += 1
        return res
```

Деревья

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

1. Имея корни двух бинарных деревьев p и q, напишите функцию, проверяющую, совпадают ли они или нет. Два бинарных дерева считаются одинаковыми, если они структурно идентичны, а узлы имеют одинаковое значение.

```
def isSameTree(p: Optional[TreeNode], q: Optional[TreeNode]) -> bool:
        def check(p, q):
            if not p and not q:
                return True
            if not q or not p:
                return False
            if p.val != q.val:
                return False
            return True

        deq = deque([(p, q),])

        while deq:
            p, q = deq.popleft()
            if not check(p, q):
                return False

            if p:
                deq.append((p.left, q.left))
                deq.append((p.right, q.right))

        return True
```

1. Дан корень бинарного дерева, проверьте, является ли он зеркалом самого себя (т.е. Симметричным относительно своего центра).

```
def isSymmetric(root: Optional[TreeNode]) -> bool:
        if not root:
            return True
        return check_leftright(root.left, root.right)

def check_leftright(node1, node2):
        if not node1 and not node2:
            return True
        elif node1 and node2 and  node1.val == node2.val:
            return check_leftright(node1.left, node2.right) and
check_leftright(node1.right, node2.left)
        else:
            return False
```

1. Учитывая бинарное дерево, определите, сбалансировано ли оно по высоте. Для этой задачи сбалансированное по высоте бинарное дерево определяется как: бинарное дерево, в котором левое и правое поддеревья каждого узла отличаются по высоте не более чем на 1.

```
# time: O(n) / space: O(1)
def isBalanced(root: Optional[TreeNode]) -> bool:
        def depth(root):
            if not root:
                return 0

            left_depth = depth(root.left)
            right_depth = depth(root.right)

            if left_depth == -1 or right_depth == -1 or abs(left_depth
- right_depth) > 1:
                return -1

            return max(left_depth, right_depth) + 1

        return depth(root) != -1
```

1. Учитывая корень двоичного дерева и целое число targetSum, вернуть все пути от корня к листу, где сумма значений узлов в пути равна targetSum. Каждый путь должен быть возвращен в виде списка значений узлов, а не ссылок на узлы. Путь от корня к листу — это путь, начинающийся от корня и заканчивающийся в любом конечном узле. Лист — это узел без потомков.

```
def pathSum(root: Optional[TreeNode], targetSum: int) ->
List[List[int]]:
        res = []

        def dfs(root, targetSum, path):
            if not root:
                return
            if root.left is None and root.right is None and targetSum
- root.val == 0:
                res.append(path + [root.val])

            dfs(root.left, targetSum - root.val, path + [root.val])
            dfs(root.right, targetSum - root.val, path + [root.val])

        dfs(root, targetSum, [])
        return res
```

Жадные алгоритмы

1. Вам дан массив цен, где prices[i] — цена данной акции на i-й день. Вы хотите максимизировать свою прибыль, выбрав один день для покупки одной акции и выбрав другой день в будущем для продажи этой акции. Верните максимальную прибыль, которую вы можете получить от этой сделки. Если вы не можете получить никакой прибыли, верните 0.

```
def maxProfit(prices: List[int]) -> int:
        min_price = prices[0]
        max_profit = 0

        for i in range(1, len(prices)):
            if min_price > prices[i]:
                min_price = prices[i]
            else:
                if prices[i] - min_price > max_profit:
                    max_profit = prices[i] - min_price

        return max_profit
```

1. Вам дан целочисленный массив цен, где prices[i] — цена данной акции на i-й день. Каждый день вы можете решить купить и/или продать акции. В любой момент времени вы можете владеть не более чем одной акцией. Однако вы можете купить его и тут же продать в тот же день. Найдите и верните максимальную прибыль, которую вы можете получить.

```
def maxProfit(prices: List[int]) -> int:
        if len(prices) == 0:
            return 0
        i = 0
        profit = 0
        while i < len(prices) - 1:
            while i < len(prices) - 1 and prices[i] >= prices[i + 1]:
                i += 1
            valley = prices[i]
            while i < len(prices) - 1 and prices[i] <= prices[i + 1]:
                i += 1
            peak = prices[i]
            profit += peak - valley
        return profit
```

1. Вам дан массив цен, где prices[i] — это цена данной акции на i-й день, а целое число представляет комиссию за транзакцию. Найдите максимальную прибыль, которую вы можете получить. Вы можете совершать столько транзакций, сколько хотите, но вам необходимо платить комиссию за транзакцию за каждую транзакцию.

Примечание. Вы не можете совершать несколько транзакций одновременно (т. е. вы должны продать акции, прежде чем покупать их снова).

```
def maxProfit(prices: List[int], fee: int) -> int:
        buying, selling = 0, -prices[0]
        for i in range(1, len(prices)):
            buying = max(buying, selling + prices[i] - fee)
            selling = max(selling, buying - prices[i])
        return buying
```

1. Вам дан массив цен, где prices[i] — цена данной акции на i-й день. Найдите максимальную прибыль, которую вы можете получить. Вы можете совершать сколько угодно транзакций (т. е. покупать одну и продавать одну акцию несколько раз) со следующими ограничениями: После того, как вы продали свои акции, вы не можете купить акции на следующий день (т. е. один день восстановления).

Примечание. Вы не можете совершать несколько транзакций одновременно (т. е. вы должны продать акции, прежде чем покупать их снова).

```
def maxProfit(prices: List[int]) -> int:
        n = len(prices)
        if n <= 1:
            return 0
        buy = -prices[0]
        sell = 0
        cooldown = 0
        for i in range(1, n):
            pre_buy = buy
            pre_sell = sell
            pre_cooldown = cooldown
            buy = max(pre_buy, pre_cooldown-prices[i])
            sell = max(pre_sell, prices[i]+pre_buy)
            cooldown = max(pre_cooldown,pre_sell)
        return max(sell,cooldown)
```