Алгоритмы и их реализация
Связный список

Цикл в связном списке

Имея связный список, определите, есть ли в нем цикл.
Продолжение: Можете ли вы решить эту задачу, не используя дополнительную память?
URL: https://leetcode.com/problems/linked-list-cycle/

```
# 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
```

Перевернуть связный список

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

```
# 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
```

Деревья

Сериализация - это процесс преобразования структуры данных или объекта в последовательность битов, чтобы они могли быть сохранены в файле или буфере памяти или переданы по каналу сетевого подключения для последующего восстановления в той же или другой среде.

Разработайте алгоритм для сериализации и десериализации бинарного дерева. Нет никаких ограничений на то, как должен работать ваш алгоритм сериализации/десериализации. Вам просто нужно убедиться, что двоичное дерево может быть преобразовано в строку, и эта строка может быть десериализована в исходную древовидную структуру.

Например, вы можете сериализовать следующее дерево

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

или

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

Вам не обязательно следовать этому формату, поэтому, пожалуйста, проявите творческий подход и придумайте другие подходы самостоятельно.

Примечание: Не используйте переменные-члены класса/ глобальные/ статические переменные для хранения состояний. Ваши алгоритмы сериализации и десериализации должны быть без учета состояния.

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
```

```
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))
```

Прямой обход дерева

Для заданного двоичного дерева верните значения его узлов.

Например: Для заданного двоичного дерева {1,#,2,3}, 1 \ 2 / 3 верните [1,2,3].

Примечание: Рекурсивное решение тривиально, не могли бы вы сделать это итеративно?

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

```
# 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 итератор

Ваш итератор будет инициализирован с корневого узла дерева.

Вызов next() возвращает следующий наименьший номер по бинарному дереву.

Примечание: функции next() и hasNext() должны выполняться в среднем за O(1) время и использовать O(h) памяти, где h - высота дерева.

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

```
# 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())
```

Графы

Количество связанных компонентов в неориентированном графе

Учитывая n узлов, помеченных от 0 до n - 1, и список неориентированных ребер (каждое ребро представляет собой пару узлов), напишите функцию для определения количества связанных компонентов в неориентированной графе.

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

```
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]
```

```
    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]
```

```
        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))
```


```
                    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))
```

Расписание курсов

Всего вам необходимо пройти n курсов, обозначенных от 0 до n - 1.
Для некоторых курсов могут быть предусмотрены предварительные условия, например, чтобы пройти курс 0, вы должны сначала пройти курс 1, который выражается в виде пары: [0,1]

Учитывая общее количество курсов и список обязательных пар, возможно ли для вас закончить все курсы?

Например:

2, [[1,0]] Всего необходимо пройти 2 курса. Чтобы пройти курс 1, вы должны пройти курс 0. Так что это возможно.

2, [[1,0],[0,1]] Всего нужно пройти 2 курса. Чтобы пройти курс 1, вы должны были закончить курс 0, а чтобы пройти курс 0, вы также должны были закончить курс 1. Таким образом, это невозможно.

Примечание: Предварительные входные данные - это граф, представленный списком ребер, а не матрицами смежности.

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

```
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
```

```
    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):
```

```
        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)
```

```
        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":
```

```
                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]]))
```

Кучи

Объединить K отсортированных связных списков

Объедините k отсортированных связных списков и верните их в виде одного отсортированного списка. Проанализируйте и опишите его сложность.

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

```
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
```

K-й по величине элемент в массиве

Найдите k-й по величине элемент в неотсортированном массиве. Обратите внимание, что это k-й по величине элемент в отсортированном порядке, а не k-й отдельный элемент.

Например, [3,2,1,5,6,4] и k = 2, верните 5.

Примечание: 1 ≤ k ≤ длина массива

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

```
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]
```

Массивы

2 Суммы II

Дан массив целых чисел, который уже отсортирован в порядке возрастания, найдите два числа, которые в сумме дают определенное число.

Функция twoSum должна возвращать индексы двух чисел, которые в сумме
дают целевое значение, где индекс 1 должен быть меньше индекса 2. Пожалуйста, обратите внимание, что возвращенные вами ответы (как с индексом1, так и с индексом2) не основаны на нулевом значении.
Вы можете допустить, что каждый ввод будет содержать ровно одно решение.
Входные данные: числа={2, 7, 11, 15}, цель=9 Выходные данные: индекс1=1, индекс2=2

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

```
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 Суммы III

Спроектируйте и реализуйте класс twoSum. Он должен поддерживать следующие операции: add и find.

add - Добавить число во внутреннюю структуру данных. find - Определить, существует ли какая-либо пара чисел, сумма которых равна значению.

Например, добавить(1); добавить(3); добавить(5); найти(4) -> истина, найти(7) -> ложь

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

```
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))
```

Строки

Анаграмма

Даны две строки s и t, напишите функцию, которая определит, является ли t анаграммой s.
Например:
s = "анаграмма"
t = “мамнагара"
функция возвращает true

s = "крыса"
t = "автомобиль"
функция возвращает false.

Примечание: Строка содержит только строчные буквы.

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

```
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
```

Палиндром

Получив строку, определите, является ли она палиндромом, учитывая только буквенно-цифровые символы и игнорируя регистры.

Например, "Бел хлеб" - это палиндром. "Участвовать в автомобильных гонках" не является палиндромом.

Примечание: Вы не задумывались о том, что строка может быть пустой? Этот вопрос стоит задать во время собеседования.

Для решения этой задачи мы определяем пустую строку как допустимый палиндром.

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

```
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
```

Управление битами

Сумма двух целых чисел

Вычислите сумму двух целых чисел a и b, но вам не разрешается использовать операторы + и -.

Пример: если a = 1 и b = 2, верните значение 3.

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

```
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)
```

Число

В заданном массиве целых чисел каждый элемент отображается дважды, за исключением одного. Найдите этот единственный элемент.

Примечание: Ваш алгоритм должен иметь линейную сложность во время выполнения.

Могли бы вы реализовать его без использования дополнительной памяти?

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

```
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
```

Математика

Обратное целое число

Обратные цифры целого числа.

Пример 1: x = 123, возвращает 321 Пример 2: x = -123, возвращает -321

Подумайте над этим:
Если последняя цифра целого числа равна 0, каким должен быть результат?
например, в таких случаях, как 10,100.

Для решения этой задачи предположим, что ваша функция возвращает 0 при переполнении перевернутого целого числа.

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
```

Число-палиндром

Определите, является ли целое число палиндромом. Сделайте это без дополнительного пробела.

Несколько советов:
Могут ли отрицательные целые числа быть палиндромами? (например, -1)

Если вы собираетесь преобразовать целое число в строку, обратите внимание на ограничение использования дополнительного пробела.

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

```
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
```

Матрица

Поворот изображения

Дана двумерная матрица n x n, представляющая изображение.

Поверните изображение на 90 градусов (по часовой стрелке).

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

```
class Solution(object):
    def rotate(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: void Do not return anything, modify matrix in-pl
ace 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][fir
st]
                    matrix[last - offset][first] = matrix[last][
last - offset]
                    matrix[last][last - offset] = matrix[i][last
]
                    matrix[i][last] = top
```

Установить нулевые значения матрицы

При заданной матрице m x n, если элемент равен 0, установите для всей его строки и столбца значение 0

Продолжение: Вы использовали дополнительную память? Простое решение с использованием O(mn) памяти, вероятно, плохая идея. Простое усовершенствование использует O (m + n) памяти, но все равно это не лучшее решение. Не могли бы вы предложить решение с постоянным использованием памяти?

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

```
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
```

Решение с постоянным использованием памяти:

```
class Solution(object):
    def setZeroes(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: void Do not return anything, modify matrix in-pl
ace 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
```