**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Ордена Трудового Красного Знамени**

**федеральное государственное бюджетное**

**образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра “Математическая кибернетика и информационные технологии”

Курсовая работа

по дисциплине: «Структуры и алгоритмы обработки данных»

Выполнил: студент группы БФИ 2001

Лушин Е.А.

Проверил: Мкртчян Г.М.

Вариант №13

Москва 2022

**Оглавление**

[Цель работы 3](#_Toc102845639)

[Задание №1 3](#_Toc102845640)

[Условие задания 3](#_Toc102845641)

[Решение задания 4](#_Toc102845642)

[Задание №2 6](#_Toc102845643)

[Условие задания 6](#_Toc102845644)

[Решение задания 7](#_Toc102845645)

[Задание №3 8](#_Toc102845646)

[Условие задания 8](#_Toc102845647)

[Решение задания 9](#_Toc102845648)

[Задание №4 11](#_Toc102845649)

[Условие задания 11](#_Toc102845650)

[Решение задания 12](#_Toc102845651)

[Задание №5 13](#_Toc102845652)

[Условие задания 13](#_Toc102845653)

[Решение задания 14](#_Toc102845654)

[Задание №6 15](#_Toc102845655)

[Условие задания 15](#_Toc102845656)

[Решение задания 16](#_Toc102845657)

[Задание №7 17](#_Toc102845658)

[Условие задания 17](#_Toc102845659)

[Решение задания 18](#_Toc102845660)

[Задание №8 19](#_Toc102845661)

[Условие задания 19](#_Toc102845662)

[Решение задания 20](#_Toc102845663)

[Задание №9 21](#_Toc102845664)

[Условие задания 21](#_Toc102845665)

[Решение задания 21](#_Toc102845666)

[Задание №10 22](#_Toc102845667)

[Условие задания 22](#_Toc102845668)

[Решение задания 23](#_Toc102845669)

[Вывод 24](#_Toc102845670)

# **Цель работы**

Необходимо найти решение для 10 задач в соответствии с 13 Вариантом.

# **Задание №1**

## **Условие задания**

Из n вершин. Для каждой вершины v от 0 до n - 1 он записал два числа degreev и sv, где первое число — количество вершин, смежных с вершиной v, а второе — XOR-сумма номеров вершин, смежных с v (в случае, если смежных вершин не было, он записал 0).

На следующий день Миша не смог вспомнить, какой граф у него был изначально. У Миши остались значения degreev и sv. Помогите ему найти количество ребер и сами ребра исходного графа. Гарантируется, что существует лес, которому соответствуют выписанные Мишей числа.

Входные данные В первой строке находится целое число n (1 ≤ n ≤ 216), количество вершин в графе.

В i-й из последующих строк находятся числа degreei и si (0 ≤ degreei ≤ n - 1, 0 ≤ si < 216), разделенные пробелом.

Выходные данные В первой строке выведите число m, количество ребер графа.

Далее выведите m строк, каждая из которых содержит два различных числа a и b (0 ≤ a ≤ n - 1, 0 ≤ b ≤ n - 1), соответствующие ребру (a, b).

Рёбра могут быть выведены в любом порядке; вершины одного ребра также могут быть выведены в любом порядке.

**Примеры**

**Входные данные**

3

2 3

1 0

1 0

**Выходные данные**

2

1 0

2 0

## **Решение задания**

class Deque:

def \_\_init\_\_(self):

self.items = []

def push\_right(self, item): # добавить элемент справа

self.items.append(item)

def push\_left(self, item): # добавить элемент слева

self.items.insert(0, item)

def pop\_right(self): # удалить элемент справа

return self.items.pop()

def pop\_left(self): # удалить элемент слева

return self.items.pop(0)

def top\_right(self): # просмотр правого элемента

return self.items[len(self.items) - 1]

def top\_left(self): # просмотр левого элемента

return self.items[0]

def empty(self): # проверка на пустоту дека

return self.items == []

def size(self): # количество элементов в деке

return len(self.items)

print("-----------------------------Входные данные-----------------------------")

n = int(input("Введите количество вершин: ")) # Вводим количество вершин

list\_data = [] # Инициализируем список для записи введённых значений degree и s

single\_vertex = Deque() # Инициализируем Deque

for i in range(n): # Цикл от 0 до количества вершин

degreev, sv = map(int, input("Введите degreev и sv: ").split()) # Ввод количества смежных вершин и XOR функций

if (degreev == 1): # Если количество смежных вершин = 1, то можем сразу занести в очередь

single\_vertex.push\_right(i) # Заносим в очередь

list\_data.append([degreev, sv]) # Записываем введёные значения в список

count = 0 # Инициализируем счётчик количества рёбер

edges = '' # Инициализируем переменную для вывода названий рёбер

while not single\_vertex.empty(): # Пока очередь не пуста

vertex = single\_vertex.pop\_left() # Рассматриваем вершину из очереди

degreev, sv = list\_data[vertex] # Берём записанную пару количества смежных вершин и XOR суммы для данной вершины

if degreev == 0: # Если смежных вершин нет, пропускаем вершину

continue

second\_degreev, second\_sv = list\_data[sv] # Берём значения вершины равняющейся XOR сумме

if second\_degreev == 2: # Если у неё смежных вершин 2

single\_vertex.push\_right(sv) # Добавляем в очередь

list\_data[sv] = [second\_degreev-1, second\_sv^vertex] # Удаляем ребро

edges += str(vertex)+' '+str(sv)+'\n' # Записываем найденное ребро

count += 1 # Увеличиваем счётчик рёбер

print("-----------------------------Выходные данные----------------------------")

print("Количество рёбер графа = ", count) # Выводим количество рёбер

print("Рёбра")

print(edges) # Выводим сами рёбра

На рисунке 1 представлен результат работы программы.

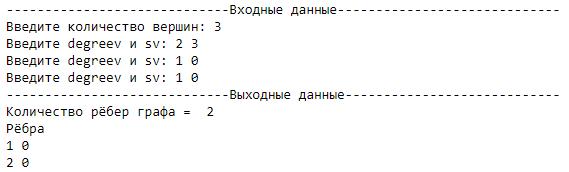


Рисунок 1 – Результат работы программы

# **Задание №2**

## **Условие задания**

Дома у Фродо собираются ночевать n хоббитов. У Фродо есть n кроватей, стоящих в ряд, и m подушек (n ≤ m). Каждому хоббиту для сна нужна кровать и хотя бы одна подушка, однако, каждый хочет как можно больше подушек. Понятно, что подушки не всегда можно поделить поровну, однако любой хоббит обидится, если у него будет меньше подушек, чем у соседа, хотя бы на две.

Фродо будет спать на k-й с краю кровати. Какое максимальное число подушек он может положить себе, чтобы у каждого хоббита была хотя бы одна подушка, все подушки были розданы хоббитам, и никто из них не обиделся?

Входные данные В единственной строке находятся три целых числа n, m и k (1 ≤ n ≤ m ≤ 109, 1 ≤ k ≤ n) — число хоббитов, число подушек и номер кровати Фродо.

Выходные данные Выведите одно число — максимальное число подушек, которые Фродо может положить себе, чтобы никто не обиделся.

**Примеры**

**Входные данные**

4 6 2

**Выходные данные**

2

## **Решение задания**

print("-----------------------------Входные данные-----------------------------")

n, m, k = map(int, input("Введите количество хоббитов, общее число подушек, № кровати Фродо, соответственно: ").split()) # Ввод ->

#-> количества хоббитов, количества подушек, номер кровати Фродо

def one\_sided(slots, x): # Функция подсчёта подушек для каждого края независимо друг от друга

if slots >= x-1: # Если кроватей больше или равно оставшимся подушкам

result = (slots - x + 1) + (x-1)\* x // 2

return result

else:

result = slots \* ( 2\* x - slots -1 ) // 2

return result

def pillows\_used(x): # Функция проверки используемых подушек

left\_slot = k -1 # Кроватей слева от Фродо

right\_slot = n-k # Кроватей справа от Фродо

res = x + one\_sided(left\_slot, x) + one\_sided(right\_slot, x) <= m # Проверяем, есть ли ещё свободные подушки или ->

#-> подушек потребовалось больше чем есть

return res

left = 1 # Инициализация переменной для левой гранциы (за границу берём подушки)

right = m # Инициализация переменной для правой границы

mid = (left + right) // 2 # Находим среднее число подушек (по сути кол-во подушек Фродо +1)

while left != right: # Пока границы проверки не пересекли друг друга

if pillows\_used(mid): # Если остаются невостребованные подушки

left = mid + 1 # То нижнюю границу нужно увеличить (увеличиваем по факту кол-во подушек у Фродо и по остаточному принципу у хоббитов)

else: # Иначе если подушек не хватило

right = mid # Уменьшаем верхнюю верхнюю границу (уменьшаем по факту кол-во подушек у Фродо и по остаточному принципу у хоббитов)

mid = (left + right) // 2 # Обновляем среднее число подушек

print("-----------------------------Выходные данные----------------------------")

print("Количество подушек у Фродо = ", mid - 1) # Выводим кол-во подушек у Фродо

На рисунке 2 представлен результат работы программы.

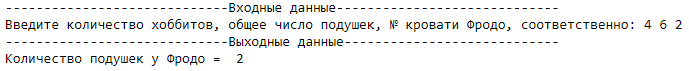


Рисунок 2 – Результат работы программы

# **Задание №3**

## **Условие задания**

Недавно Вы купили новую умную лампу с программируемыми функциями. Первым делом Вы выставили расписание на ней. Каждый день лампа будет включать питание в момент 0 и выключать питание в момент 𝑀. Более того, данная лампа позволяет Вам записать в нее программу, следуя которой, она будет менять свое состояние (состояниями считается «свет включен» и «свет выключен»). К сожалению, в лампу уже предустановлена некоторая программа.

Лампа принимает только хорошие программы. Хорошая программа может быть представлена непустым массивом 𝑎, где 0<𝑎1<𝑎2<⋯<𝑎|𝑎|<𝑀. Все 𝑎𝑖 должны быть целыми числами. Конечно, предустановленная программа является хорошей.

Лампа выполняет заданную программу 𝑎 следующим образом: в момент 0 она включает и питание, и свет. Далее, в момент 𝑎𝑖 лампа переключает свое состояние на противоположное (если свет был включен, то он выключается, и наоборот). Свет переключается моментально, т. е., например, если переключить свет в момент времени 1 и дальше ничего не делать, то итоговое время горения лампы будет равно 1. Наконец, в момент 𝑀 лампа отключает питание независимо от того, был ли свет включен.

Так как Вы не из тех, кто читает инструкции, да и написана она на неизвестном языке, Вы (методом проб и ошибок) находите единственный способ изменить предустановленную программу. Вы можете вставить не более одного числа в программу 𝑎 таким образом, что она все еще останется хорошей. Вставка может быть осуществлена в любое место, как между любой парой соседних элементов в 𝑎, так и в начало или конец массива 𝑎.

Найдите такой способ изменить программу, что суммарное время горения лампы будет максимально. Возможно, что иногда лучше оставить программу нетронутой. (Если у лампы включен свет с момента 𝑥 по момент 𝑦, тогда лампа будет гореть время, равное 𝑦−𝑥. Отрезки включенного света суммируются).

Входные данные Первая строка содержит два числа 𝑛 и 𝑀 (1≤𝑛≤105, 2≤𝑀≤109) — длина предустановленной программы 𝑎 и момент отключения питания лампы.

Вторая строка содержит 𝑛 чисел через пробел 𝑎1,𝑎2,…,𝑎𝑛 (0<𝑎1<𝑎2<⋯<𝑎𝑛<𝑀) — программа 𝑎.

Выходные данные Выведите единственное число — максимально возможное время горения, если Вы можете сделать не более одного изменения, описанного выше.

**Примеры**

**Входные данные**

3 10

4 6 7

**Выходные данные**

8

## **Решение задания**

print("-----------------------------Входные данные-----------------------------")

n, m = map(int, input("Введите длину программы и момент отключения света, соответственно: ").split()) # Ввод длины программы ->

#-> и команды на выключение

program = [0] + [int(x) for x in input("Введите числа программы a: ").split()] + [m] # Добавим к массиву программы в начало 0 ->

#-> (включение питания и света) и команды М в конец массива (выключение питания)

lamp\_off = 0 # Суммарное время не работы лампочки

lamp\_on = 0 # Суммарное время работы лампочки

final = 0 # Выигранное время

for i in range(1, n+2): # От 1 номера массива (от 0 если не считать добавленный ноль) ->

#-> до последнего элемента массива (+2 из-за того, что добавили М)

if i % 2 == 0: # Если номер элемента чётный

lamp\_off = lamp\_off + (program[i] - program[i-1]) # Прибавляем время неработающей лампочки

else:

lamp\_on = lamp\_on + (program[i] - program[i-1]) # Прибавляем время работающей лампочки

maximum = lamp\_on # Принимаем за максимум полученное время работы лампочки

for i in range(1, n+1, 2): # Проверяем пары элементов

final = final + (program[i] - program[i-1])

if ((program[i] - program[i-1] > 1) or (program[i+1] - program[i] > 1)): # Если разница между прошлым элементом или ->

#-> будущем равна 1, то максимум персчитывать не нужно ()

maximum = max(maximum, final-1 + lamp\_off) # За новый максимум берём либо прежний максимум,->

#-> либо найденное время работы лампочки + оставшееся нерабочее время лампочки на этой итерации

lamp\_off = lamp\_off - (program[i+1] - program[i]) # Пересчитываем время не работы лампочки

print("-----------------------------Выходные данные----------------------------")

print("Максимально возможное время горения света = ", maximum) # Вывод максимального времени работы лампочки

На рисунке 3 представлен результат работы программы.

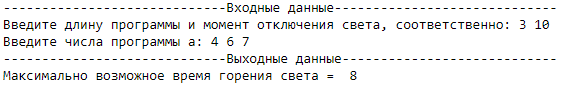


Рисунок 3 – Результат работы программы

# **Задание №4**

## **Условие задания**

У Васи есть n предметов, которые лежат в ряд. Предметы последовательно пронумерованы числами от 1 до n таким образом, что самый левый предмет имеет номер 1, самый правый — номер n. Каждый предмет имеет свой вес, i-тый из них весит wi килограмм.

Васе нужно собрать все эти предметы, однако он не будет делать этого сам, а использует своего нового робота. У робота есть две разные руки — левая и правая. Робот может последовательно выполнять действия:

Взять левой рукой самый левый из предметов, затратив на это wi · l единиц энергии (где wi — вес самого левого предмета, l — заданный параметр), если предыдущее действие было таким же (левой рукой), то робот тратит дополнительные Ql единиц энергии. Взять правой рукой самый правый из предметов, затратив на это wj · r единиц энергии (где wj — вес самого правого предмета, r — заданный параметр), если предыдущее действие было таким же (правой рукой), то робот тратит дополнительные Qr единиц энергии. Разумеется, Вася хочет запрограммировать робота так, чтобы тот потратил как можно меньше энергии. С этой задачей он и обратился за помощью к вам. Ваша задача — найти минимальное количество энергии, которое потратит робот, чтобы собрать все предметы.

Входные данные В первой строке содержится пять целых чисел n, l, r, Ql, Qr (1 ≤ n ≤ 105; 1 ≤ l, r ≤ 100; 1 ≤ Ql, Qr ≤ 104).

Во второй строке содержится n целых чисел w1, w2, ..., wn (1 ≤ wi ≤ 100).

Выходные данные В единственной строке выведите целое число — ответ на задачу.

**Примеры**

**Входные данные**

3 4 4 19 1

42 3 99

**Выходные данные**

576

## **Решение задания**

print("-----------------------------Входные данные-----------------------------")

n, L, R, QL, QR = map(int, input("Введите n, L, R, QL, QR, соответственно: ").split()) # Вводим количество предметов,

# доступных действий левой рукой, доступных действий правой рукой, штраф левой руки, штраф правой руки

W = list(map(int, input("Введите веса предметов: ").split())) # Список весов предметов

sum\_el = [0] # Инициализация массива суммированных предметов для перебора вариантов переноса предметов

for i in range(1, n + 1): # От 1 номера элемента, до последнего (из-за 0 в начале смешаем на +1)

sum\_el.append(W[i - 1] + sum\_el[i - 1]) # Заполняем список проссумированных элементов

answer = QR \* (n - 1) + sum\_el[n] \* R # Расчёт затраченной энергии при условии, что всё выполняется правой рукой

for i in range(1, n + 1): # Проходим циклом по списку просуммированых элементов

energy = L \* sum\_el[i] + R \* (sum\_el[n] - sum\_el[i]) # Расчёт энергии до штрафов (перебор рук по сути)

if i > (n - i): # Если переложили левой рукой больше чем правой

energy = energy + (i - (n - i) - 1) \* QL # Считаем штраф за это

elif (n - i) > i: # Иначе если переложили правой рукой больше чем левой

energy = energy + ((n - i) - i - 1) \* QR # Считаем штраф за это

if energy < answer: # Если расчитанная энергия меньше ранее полученного ответа, то эта энергия становится ответом

answer = energy

print("-----------------------------Выходные данные----------------------------")

print("Минимально затраченная энергия = ", answer) # Выводим полученную энергию

print("-----------------------------Выходные данные----------------------------")

print("Максимально возможное время горения света = ", maximum) # Вывод максимального времени работы лампочки

На рисунке 4 представлен результат работы программы.

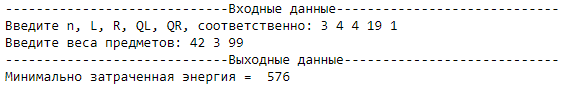


Рисунок 4 – Результат работы программы

# **Задание №5**

## **Условие задания**

Будем называть неориентированный граф из n вершин p-интересным, если выполнены условия:

граф содержит ровно 2n + p ребер; граф не содержит петель и кратных ребер; для любого целого k (1 ≤ k ≤ n) любой подграф, состоящий из k вершин, содержит не более 2k + p ребер. Подграфом графа будем называть некоторое множество вершин графа и некоторое множество ребер графа. Причем множество ребер должно удовлетворять условию: оба конца каждого ребра из множества должны принадлежать выбранному множеству вершин.

Ваша задача отыскать p-интересный граф, состоящий из n вершин.

Входные данные В первой строке задано единственное целое число t (1 ≤ t ≤ 5) — количество тестовых данных. В следующих t строках задано по два целых числа: n, p (5 ≤ n ≤ 24; p ≥ 0; ) — количество вершин в графе и параметр интересности для соответствующего теста.

Гарантируется, что искомый граф существует.

Выходные данные Для каждого из t тестов выведите 2n + p строк, содержащих описание ребер p-интересного графа: i-я строка должна содержать два целых числа через пробел ai, bi (1 ≤ ai, bi ≤ n; ai ≠ bi) — две вершины, соединенные ребром в результирующем графе. Считайте, что вершины графа пронумерованы целыми числами от 1 до n.

Ответы для тестов выводите в том порядке, в котором тесты заданы во входных данных. Если существует несколько решений, разрешается вывести любое из них.

**Примеры**

**Входные данные**

1

6 0

**Выходные данные**

1 2

1 3

1 4

1 5

1 6

2 3

2 4

2 5

2 6

3 4

3 5

3 6

## **Решение задания**

print("-----------------------------Входные данные-----------------------------")

t = int(input("Введите количество тестовых данных: ")) # Вводим количество тестовых данных

for x in range(t): # Цикл от 0 до введёного количества тестовых данных

if x != 0:

print("-----------------------------Входные данные-----------------------------")

n, p = map(int, input("Введите количество вершин и параметр интересности соответственно: ").split()) # Вводим количество ->

#-> вершин в графе и параметр интересности для соответствующего теста

print("-----------------------------Выходные данные----------------------------")

print("Вывод тестового набора данных №", x+1)

end\_output = 2\*n + p

count = 0 # Инициализируем счётчик

for i in range(n): # Цикл от 0 до количества вершин

for j in range(i+1, n): # Вложенный цикл для вывода рёбер графа

print(i+1, j+1) # Выводим две вершины

count = count + 1 # Увеличиваем счётчик

if (count == end\_output): # Если достигли количества строк из условия о выводе 2\*n + p строк

break # Завершаем выполнение программы

if count == end\_output: # Если достигли количества строк из условия о выводе 2\*n + p строк

break # Завершаем выполнение программы

На рисунке 5 представлен результат работы программы.

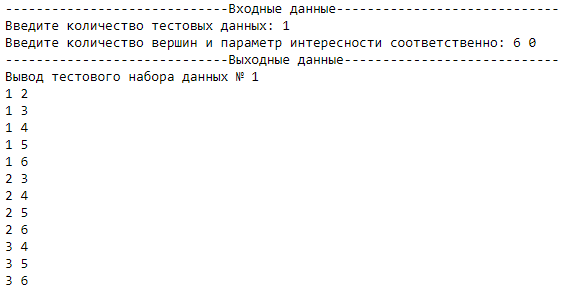


Рисунок 5 – Результат работы программы

# **Задание №6**

## **Условие задания**

Недавно Дима познакомился с Сашей в филателистическом магазине, и с тех пор они собирают монеты вместе. Их любимое занятие — сортировать коллекции монет. Саше нравится порядок, поэтому он хочет, чтобы монеты были расположены в ряд, причём сначала шли монеты, вышедшие из обращения, а потом монеты, всё ещё находящиеся в обращении.

Для упорядочивания монет Дима использует алгоритм, один шаг которого выглядит следующим образом:

Дима просматривает все монеты слева направо; если он видит, что i-я монета ещё в ходу, а (i + 1)-я уже вышла из обращения, то он меняет эти две монеты местами, и продолжает смотреть на монеты дальше, начиная с (i + 1)-й. Дима повторяет этот шаг до тех пор, пока не окажется, что на очередном шаге не произошло ни одного обмена. Сложностью упорядочивания Дима называет количество шагов, которые ему требуются в соответствии с процедурой, описанной выше, то есть, количество раз, которое он будет начинать просматривать монеты с начала. В частности, для уже упорядоченной исходной последовательности монет сложность упорядочивания равна единице.

Сегодня Саша в очередной раз позвал Диму в гости, чтобы предложить ему следующую игру. Сначала он выложил в ряд перед Димой n монет, все из которых вышли из обращения. Затем Саша n раз выбирает какую-то из монет, которая вышла из обращения, и заменяет её на монету, которая находится в обращении. В ходе этого процесса Саша постоянно интересуется у Димы, какова на данный момент сложность упорядочивания последовательности.

Задачу усложняет тот факт, что Диме нельзя трогать монеты, и поэтому определять сложность упорядочивания ему приходится в уме. Помогите Диме справиться с этим заданием.

Входные данные В первой строке задано целое число n (1 ≤ n ≤ 300 000) — количество монет, которые Саша выложил в ряд перед Димой.

Следующая строка содержит n целых различных чисел p1, p2, ..., pn (1 ≤ pi ≤ n) — позиции монет, если смотреть слева направо, которые Саша меняет на монеты, находящиеся в ходу. Сначала Саша заменяет монету, находящуюся на позиции p1, затем монету, находящуюся на позиции p2 и так далее.

Выходные данные Выведите n + 1 чисел a0, a1, ..., an, где a0 — сложность упорядочивания последовательности в начале, a1 — сложность упорядочивания после замены одной монеты и так далее.

**Примеры**

**Входные данные**

4

1 3 4 2

**Выходные данные**

1 2 3 2 1

## **Решение задания**

print("-----------------------------Входные данные-----------------------------")

n = int(input("Введите количество монет: ")) # Вводим количество монет

arr = list(map(int, input("Введите позиции монет: ").split())) # Вводим позиции монет

pos = n # Инициализируем переменную для отслеживания позиции монеты

a = [0 for i in range(n+1)] # Инициализируем список который будет отслеживать замену монет ->

#-> (0 - вышла из обращения, 1 - осталась в обращении)

res = 1 # Инициализируем переменную для подсчёта сложности упорядочивания последовательности

ans = [1] # Инициализируем список в котором будут записаны сложности упорядочивания ->

#-> (сразу заносим 1, так как из условия сказано, что сложность упорядоченной последовательности равна 1)

for x in arr: # Проходим по введённой последовательности позиций монет

a[x] = 1 # Помечаем текущую монету как заменённую на используемую в обращении

res = res + 1 # Повышаем сложность упорядоченвиания последовательности

while a[pos] == 1: # Пока текущая монета на текущей позиции используется в обращении ->

#-> (пока последняя монета остаётся вышедшей из обращения, сложность лишь растёт)

pos = pos - 1 # Просматриваем, была ли заменена монета слева ->

#-> (чем ближе к началу мы встретим вышедшую монету из обращения, тем меньше будет сложность упорядочивания)

res = res - 1 # Уменьшаем сложность упорядочивания

ans.append(res) # Добавляем сложность в список сложностей упорядочивания

print("-----------------------------Выходные данные----------------------------")

print("Сложность упорядочивания последовательности после замены монеты:", ' '.join(map(str, ans))) # Выводим ответ

На рисунке 6 представлен результат работы программы.

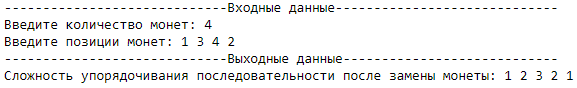


Рисунок 6 – Результат работы программы

# **Задание №7**

## **Условие задания**

Маленький Слоник очень любит суммы на промежутках.

В этот раз у него есть пара целых чисел l и r (l ≤ r). Маленькому Слонику нужно найти количество таких целых чисел x (l ≤ x ≤ r), что первая цифра числа x равна последней (в десятичном представлении). Например, такие числа как 101, 477474 или 9 он будет включать в ответ, в то время как 47, 253 или 1020 — нет.

Помогите ему, посчитайте описанное количество чисел x для заданной пары l и r.

Входные данные В единственной строке задана пара целых чисел l и r (1 ≤ l ≤ r ≤ 1018) — границы промежутка.

Пожалуйста, не используйте спецификатор %lld для чтения или записи 64-х битовых чисел на С++. Рекомендуется использовать потоки cin, cout или спецификатор %I64d.

Выходные данные В единственной строке выведите одно целое число — ответ на задачу.

**Примеры**

**Входные данные**

2 47

**Выходные данные**

12

## **Решение задания**

def find(s): # функция поиска кол-ва подходящих чисел

n = len(s) # Вычисляем длину числа

x = int(s) # Переводим в целочисленный тип данных

if (x < 10): # Если число меньше 10

return x # То нам достаточно вернуть его (так как будет достаточно просто вычесть это число из количества подходящих нам чисел)

ans = x // 10 - 1 # Если число больше 10, то считываем количество десятков ->

#-> (-1 из-за того, что первая цифра числа может быть не равна последней)

# (также нам не важно, число состоит только из десятков или ещё сотен или даже тысяч, в них всех будет не более)

ans = ans + 9 # Прибавляем 9 так как до 10 нам подходят все числа, а от лишних при вычитании счётчика первого числа избавимся

if (s[0] <= s[n-1]): # Если первая цифра числа меньше или равна последней, то включаем в подсчёт и текущею десятку

ans = ans + 1 # Поэтому к ответу прибавляем ранее отнятую единицу

return ans # Возвращаем число подходящих по условию чисел

print("-----------------------------Входные данные-----------------------------")

l, r = map(int, input("Введите пару чисел: ").split()) # Вводим пару чисел

l = l - 1 # Вычитаем из левой границы сразу 1 (так как в подсчёте границы включены)

l = str(l) # Переводим в строковый тип данных

r = str(r) # Переводим в строковый тип данных

count1 = find(l) # Инициализируем счётчик для первого числа

count2 = find(r) # Инициализируем счётчик для второго числа

print("-----------------------------Выходные данные----------------------------")

print("Ответ на задачу:", count2 - count1) # Считаем по разнице сколько чисел в промежутке нам подходит

На рисунке 7 представлен результат работы программы.

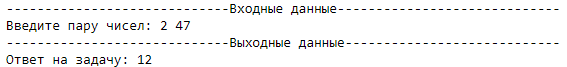


Рисунок 7 – Результат работы программы

# **Задание №8**

## **Условие задания**

После возвращения из армии Макес получил в подарок массив a, состоящий из n целых положительных чисел. Так как он очень давно не решал задачи, его заинтересовал следующий вопрос: сколько существует таких упорядоченных троек (i,  j,  k), что i < j < k, а ai·aj·ak — минимально. Помогите ему в этом!

Входные данные В первой строке входных данных задано целое положительное число n (3 ≤ n ≤ 105) — количество элементов в массиве a. Во второй строке задано n положительных целых чисел ai (1 ≤ ai ≤ 109) — элементы массива.

Выходные данные Выведите единственное число — количество упорядоченных троек (i,  j,  k), что i,  j и k — попарно различны, а ai·aj·ak — минимально.

**Примеры**

**Входные данные**

4

1 1 1 1

**Выходные данные**

4

## **Решение задания**

print("-----------------------------Входные данные-----------------------------")

n = int(input("Введите количество элементов в массиве: ")) # Вводим количество элементов в массиве

massiv\_a = list(map(int, input("Введите элементы массива: ").split())) # Вводим элементы массива

massiv\_a = sorted(massiv\_a) # Сортируем элементы массива (так как нас интерисуют 3 минимальных числа)

x, y, z = massiv\_a[0], massiv\_a[1], massiv\_a[2] # Присваиваем переменным минимальные элементы

print("-----------------------------Выходные данные----------------------------")

if (x < y < z): # Если они все различны, то число упорядоченных троек равняется количеству максимального числа из минимальных, в общем массиве чисел

print("Число упорядоченных троек =", massiv\_a.count(z)) # Выводим количество этих чисел

if ((x < y) and (y == z)): # Если различно лишь первое число, а остальные два равны

sum\_element = massiv\_a.count(z) # То считаем количество максимального числа из минимальных, в общем массиве чисел

print("Число упорядоченных троек =", (sum\_element \* (sum\_element-1)) // 2) # И для вывода необходимо воспользоваться формулой суммой n-1 числа

if ((x == y) and (y < z)): # Если различно лишь последнее число, а остальные два равны, то число упорядоченных троек равняется ->

#-> количеству максимального числа из минимальных, в общем массиве чисел

print("Число упорядоченных троек =", massiv\_a.count(z)) # Выводим количество этих чисел

if ((x == y) and (y == z)): # Если нои все равны

sum\_element = massiv\_a.count(z) # То считаем количество одинаковых чисел, в общем массиве чисел

print("Число упорядоченных троек =", (sum\_element \* (sum\_element-1) \* (sum\_element-2)) // 6) # И для вывода необходимо воспользоваться формулой ->

#-> сочетаний из n элементов по m (из количества найденных элементов по 3)

На рисунке 8 представлен результат работы программы.

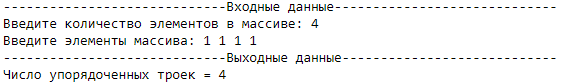


Рисунок 8 – Результат работы программы

# **Задание №9**

## **Условие задания**

Дана строка S, состоящая из N символов. Требуется найти количество упорядоченных пар целых чисел i и j таких, что:

1. 1 ≤ i, j ≤ N
2. S[i] = S[j], то есть i-ый символ строки S равен j-ому.

Входные данные Единственная строка входа содержит S, состоящую из строчных букв латинского алфавита и цифр. Гарантируется, что строка S непуста и её длина не превосходит 105.

Выходные данные Выведите одно число — количество пар i и j с требуемым свойством. Пары (x, y) и (y, x) следует считать различными, т. е. считаются упорядоченные пары.

**Примеры**

**Входные данные**

great10

**Выходные данные**

7

**Входные данные**

aaaaaaaaaa

**Выходные данные**

100

## **Решение задания**

print("-----------------------------Входные данные-----------------------------")

a=list(input("Введите строку: ")) # Вводим строку

b=[] # Инициализируем список для записи количиства повторений элементов из введённой строки

for i in set(a): # Проходим циклом по множеству (коллекции без повторяющихся элементов)

b.append(a.count(i)) # Добавляем в список количество повторений для каждого элемента

summ = 0 # Инициализируем переменную для вывода найденного количества пар

for i in b: # Проходим по списку количества повторений элементов

count = 0 # Обнуляем счётчик для каждого элемента

count = i\*\*2 # Количество пар будет равна квадрату числа повторений элемента

summ = summ + count # Прибавляем найденное количество пар текущей переменной, к общему числу пар

print("-----------------------------Выходные данные----------------------------")

print("Количество пар равно =", summ) # Выводим общее количество пар

На рисунке 9 представлен результат работы программы.

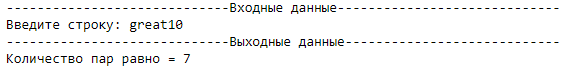


Рисунок 9 – Результат работы программы

# **Задание №10**

## **Условие задания**

Алиса и Боб играют в игру на прямой с 𝑛 ячейками. Есть 𝑛 ячеек, пронумерованных от 1 до 𝑛. Для каждого 𝑖 от 1 до 𝑛−1 ячейки 𝑖 и 𝑖+1 являются смежными.

У Алисы изначально есть фишка в какой-то ячейке на прямой, а Боб пытается угадать, где она находится.

Боб спрашивает последовательность номеров ячеек в таком порядке: 𝑥1,𝑥2,…,𝑥𝑘. В 𝑖-м вопросе Боб спрашивает Алису, находится ли ее фишка в ячейке 𝑥𝑖. То есть, Алиса ответит либо «нет», либо «да» на каждый вопрос Боба.

Не более одного раза в этом процессе, до или после ответа на вопрос, Алиса может переместить свою фишку из своей текущей ячейки в некоторую смежную ячейку. Алиса действует так, чтобы она могла ответить «нет» на все вопросы Боба.

Обратите внимание, что Алиса даже может переместить свою фигурку, прежде чем ответить на первый вопрос или после того, как ответит на последний вопрос. Алиса также может вообще не передвигать ее.

Вам дано число 𝑛 и вопросы Боба 𝑥1,…,𝑥𝑘. Вы хотели бы посчитать количество сценариев, которые позволяют Алисе ответить «нет» на все вопросы Боба.

Пусть (𝑎,𝑏) обозначает сценарий, в котором Алиса начинает в ячейке 𝑎 и заканчивает в ячейке 𝑏. Два сценария (𝑎𝑖,𝑏𝑖) и (𝑎𝑗,𝑏𝑗) различны, если 𝑎𝑖≠𝑎𝑗 или 𝑏𝑖≠𝑏𝑗.

Входные данные Первая строка содержит два целых числа 𝑛 и 𝑘 (1≤𝑛,𝑘≤105) — количество ячеек и количество вопросов Боба.

Вторая строка содержит 𝑘 целых чисел 𝑥1,𝑥2,…,𝑥𝑘 (1≤𝑥𝑖≤𝑛) — вопросы Боба.

Выходные данные Выведите единственное целое число — количество сценариев, которые позволяют Алисе ответить «нет» на все вопросы Боба.

**Примеры**

**Входные данные**

5 3

5 1 4

**Выходные данные**

9

## **Решение задания**

print("-----------------------------Входные данные-----------------------------")

n, k = map(int, input("Введите количество ячеек и вопросов Боба, соответственно: ").split()) # Вводим количество ячеек и количество вопросов Боба

x = list(map(int, input("Введите номера ячеек о которых спрашивает Боб: ").split())) # Вводим номера ячеек о которых спрашивает Боб

before = [k+1 for \_\_ in range(n+1)] # Инициализируем список ячеек перед вопросом Боба

after = [-1 for \_\_ in range(n+1)] # Инициализируем список ячеек после вопроса Боба

for i in range(k): # Проходим циклом по количеству вопросов

before[x[i]] = min(before[x[i]], i) # Изменяем значения ячеек на номера вопросов Боба, чтобы сразу определить безопасные ячейки

after[x[i]] = max(after[x[i]], i) # Изменяем значения ячеек на номера вопросов Боба, чтобы сразу определить безопасные ячейки

count = 0 # Инициализируем счётчик

for i in range(1, n+1): # Проходим по ячейкам

if after[i] == -1: # Если после вопроса можем остаться тут

count = count + 1 # Прибавляем стратегию

if (i+1 <= n): # Пока не просмотрим все ячейки как стартовую позицию

if (after[i] < before[i+1]): # Если после вопроса можем сменить позицию ->

#-> (Если Боб промахнулся с позицией о фишке и мы можем перенести фишку в ячейку, про которую он теперь не спросит)

count = count + 1 # Прибавляем стратегию

if (after[i+1] < before[i]): # Если перед вопросом можем сменить позицию ->

#-> (Если мы не перенесли ранее фишку и Боб спрашивает про позицию где она лежит, но у нас есть возможность её перенести)

count = count + 1 # Прибавляем стратегию

print("-----------------------------Выходные данные----------------------------")

print (count) # Выводим количество стратегий

На рисунке 10 представлен результат работы программы.

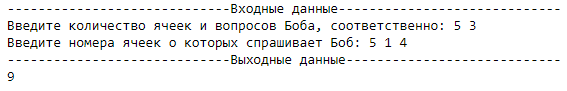


Рисунок 10 – Результат работы программы

# **Вывод**

В ходе выполнения данной курсовой работы были разработаны программы для решения 10 задач в соответствии с условием 13 Варианта.