**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»

на тему:

«Решение задач»

Выполнил: студент группы БВТ2005

Никитин Степан Романович

Руководитель: Симонов С. Е.

Москва, 2022

**Содержание**

[Задача №1 3](#_Toc114186963)

[Задача №2 4](#_Toc114186964)

[Задача №3 7](#_Toc114186965)

[Задача №5 11](#_Toc114186966)

[Задача №6 13](#_Toc114186967)

[Задача №7 15](#_Toc114186968)

[Задача №8 17](#_Toc114186969)

[Задача №9 19](#_Toc114186970)

[Задача №10 21](#_Toc114186971)

[Вывод 23](#_Toc114186972)

Задача №1

На уроке геометрии Вася узнал, что треугольник называется прямоугольным, если он невырожденный, а один из его углов прямой. Вася тут же захотел нарисовать такой треугольник: он отметил на листке бумаги 3 точки с целочисленными координатами, соединил их отрезками прямых и показал Пете. Но тот сказал, что васин треугольник не прямоугольный, а почти прямоугольный: сам треугольник не является прямоугольным, но можно подвинуть одну из точек на расстояние ровно 1 так, чтобы координаты остались целыми, и треугольник стал прямоугольным. Вася просит вас помочь ему выяснить, не обманывает ли его Петя. По координатам вершин треугольника определите, является ли он прямоугольным, почти прямоугольным, или же не является ни тем, ни другим.

Входные данные

В первой строке входного файла через пробел записано 6 целых чисел x1, y1, x2, y2, x3, y3 — координаты вершин треугольника. Все координаты целые и не превосходят по модулю 100. Гарантируется, что треугольник невырожденный, т. е. его площадь не равна нулю.

Выходные данные

Если заданный треугольник является прямоугольным, выведите RIGHT, если он является почти прямоугольным, выведите ALMOST, и если он не является ни тем, ни другим, выведите NEITHER.

**Решение**

**import** sys

**def** pythagorean\_theorem(xi, yi, xj, yj):

""" Метод для вычисления дистанции от точки до точки по координатам, то есть сторону """

**return** (xj **-** xi) **\*\*** 2 **+** (yj **-** yi) **\*\*** 2

**def** triangle(x1, y1, x2, y2, x3, y3):

st **=** pythagorean\_theorem(x1, y1, x2, y2)

nd **=** pythagorean\_theorem(x1, y1, x3, y3)

rd **=** pythagorean\_theorem(x2, y2, x3, y3)

arr **=** [st, nd, rd]

arr**.**sort()

**if** arr[0] **+** arr[1] **==** arr[2] **and** arr[0] **\*** arr[1] **\*** arr[2]: *# and Проверка "нулевой" стороны*

**return** **True**

**else**:

**return** **False**

print('-----------------------------------------\n', 'Все данные вводить строго через пробел!',

'\n-----------------------------------------\n')

x1, y1, x2, y2, x3, y3 **=** list(map(int, input('Введите координаты вершин треугольника: ')**.**split()))

num\_arr **=** [x1, y1, x2, y2, x3, y3]

**for** num **in** num\_arr:

**if** num **not** **in** range(**-**100, 101):

sys**.**exit(f'Число {num} не находится в диапазоне -100 <= X <= 100')

result **=** triangle(x1, y1, x2, y2, x3, y3)

fin\_arr **=** []

print('')

**if** result:

print('Полученный результат: RIGHT')

**else**:

fin\_arr**.**append(triangle(x1 **+** 1, y1, x2, y2, x3, y3))

fin\_arr**.**append(triangle(x1 **-** 1, y1, x2, y2, x3, y3))

fin\_arr**.**append(triangle(x1, y1 **+** 1, x2, y2, x3, y3))

fin\_arr**.**append(triangle(x1, y1 **-** 1, x2, y2, x3, y3))

fin\_arr**.**append(triangle(x1, y1, x2 **+** 1, y2, x3, y3))

fin\_arr**.**append(triangle(x1, y1, x2 **-** 1, y2, x3, y3))

fin\_arr**.**append(triangle(x1, y1, x2, y2 **+** 1, x3, y3))

fin\_arr**.**append(triangle(x1, y1, x2, y2 **-** 1, x3, y3))

fin\_arr**.**append(triangle(x1, y1, x2, y2, x3 **+** 1, y3))

fin\_arr**.**append(triangle(x1, y1, x2, y2, x3 **-** 1, y3))

fin\_arr**.**append(triangle(x1, y1, x2, y2, x3, y3 **+** 1))

fin\_arr**.**append(triangle(x1, y1, x2, y2, x3, y3 **-** 1))

**if** **True** **in** fin\_arr:

print('Полученный результат: ALMOST')

**else**:

print('Полученный результат: NEITHER')

Задача №2

После ужасающего лесного пожара в Берляндии была реализована программа восстановления леса, по которой были посажены N рядов по M деревьев в каждом, причем настолько ровно, что можно ввести систему координат, в которой j-тое дерево в i-м ряду будет иметь координаты (i, j). Но случилось ужасное, и едва окрепший лес загорелся. И теперь для составления плана эвакуации необходимо найти координаты дерева, которое загорится позже всех.

Возгорание произошло в K точках одновременно, т.е. в начальный момент времени загорелось K деревьев. Каждую минуту огонь перекидывается со всех горящих деревьев на те деревья, которые еще не горят и расстояние от которых до ближайшего горящего дерева равно 1.

Найдите дерево, которое загорится позже всех. Если таких деревьев несколько, выведите любое.

Входные данные

В первой строке входного файла заданы два целых числа N, M (1 ≤ N, M ≤ 2000) — размеры леса. Деревья были посажены во всех точках вида (x, y) (1 ≤ x ≤ N, 1 ≤ y ≤ M), x и y — целые числа.

Во второй строке задано одно целое число K (1 ≤ K ≤ 10) — количество деревьев, горящих в начальный момент времени.

В третьей строке задано K пар целых чисел: x1, y1, x2, y2, ..., xk, yk (1 ≤ xi ≤ N, 1 ≤ yi ≤ M) — координаты точек возгорания. Гарантируется, что все точки возгорания различны.

Выходные данные

Выведите одну строку, содержащую два целых числа x и y, записанные через пробел, — координаты дерева, которое загорится последним. Если таких деревьев несколько, выведите любое из них.

**Решение**

**import** sys

**def** burning\_trees(n, m, burn):

pairs **=** []

**for** i **in** range(0, len(burn), 2): *# Цикл для перекидывания огня по координатам*

x **=** burn[i]

y **=** burn[i **+** 1]

pairs**.**append((x, y))

last\_tree **=** (1, 1)

maxd **=** 0 *# Для вычисления макс дистанции*

square **=** n **\*** m

**for** i **in** range(1, n **+** 1):

**for** j **in** range(1, m **+** 1):

md **=** square

**for** pair **in** pairs:

x, y **=** pair

d **=** abs(i **-** x) **+** abs(j **-** y) *# Вычисление формулы дистанции по модулю*

md **=** min(md, d) *# Сравнение нынешней и минимальной дистанции*

**if** md **>** maxd:

last\_tree **=** (i, j)

maxd **=** md

**return** f'Последнее загоревшее дерево имеет координаты: {last\_tree[0]} {last\_tree[1]}'

print('-----------------------------------------\n', 'Все данные вводить строго через пробел!',

'\n-----------------------------------------\n')

n, m **=** map(int, input('Введите масштаб леса размером n x m: ')**.**split())

arr\_nm **=** [n, m]

**for** num **in** arr\_nm:

**if** num **not** **in** range(1, 2001):

sys**.**exit(f'Число {num} не находится в диапазоне 1 <= X <= 2000')

k **=** int(input('Введите количество "горящих" деревьев: '))

**if** k **not** **in** range(1, 11):

sys**.**exit(f'Число k = {k} не находится в диапазоне 1 <= X <= 10')

arr\_burn **=** []

**for** \_ **in** range(k):

xi, yi **=** map(int, input(f'Введите координаты {\_ **+** 1} "горящего" дерева: ')**.**split())

arr\_burn **+=** xi, yi

**for** el **in** range(len(arr\_burn)):

**if** el **//** 2 **==** 0:

**if** arr\_burn[el] **not** **in** range(1, n **+** 1):

sys**.**exit(f'Число {arr\_burn[el]} не находится в диапазоне 1 <= X <= {n}')

**else**:

**if** arr\_burn[el] **not** **in** range(1, m **+** 1):

sys**.**exit(f'Число {arr\_burn[el]} не находится в диапазоне 3 <= X <= {m}')

print('')

print(burning\_trees(n, m, arr\_burn))

Задача №3

Вы, наверное, слышали про задачу о дележе торта. Если два человека хотят справедливо разделить кусок торта, то один человек должен разрезать кусок пополам, а другой — выбрать, кому достанется какая половина. У Алисы и Боба есть несколько кусочков пирога, и, вместо того, чтобы разрезать каждый кусочек пополам, они договорились, что каждый кусочек будет съеден одним человеком.

Они решили разделить кусочки следующим образом. Сначала они определили порядок, в котором кусочки будут распределены. У них есть специальная фишка, называемая «решающий», изначально она находится у Боба. Пока не все кусочки распределены, тот, у кого находится фишка, дает очередной кусочек пирога по своему усмотрению одному участнику дележа, а фишку — другому. Так продолжается, пока все кусочки не будут распределены.

Так как все кусочки превосходного качества, каждый участник хочет максимизировать суммарный размер кусочков пирога, которые достанутся ему. Предполагая, что оба участника принимают решения оптимально, сколько пирога получит каждый?

Входные данные

В первой строке находится одно целое число N (1 ≤ N ≤ 50) — количество кусочков пирога.

На следующей строке находятся N целых чисел — размеры кусочков (каждый размер лежит в пределах от 1 до 100000 включительно) в том порядке, в котором они будут распределены.

Выходные данные

Выведите два целых числа. Сначала выведите суммарный размер кусочков, которые достанутся Алисе, а затем суммарный размер кусочков, которые достанутся Бобу, если оба участника действуют оптимально.

**Решение**

**import** sys

**def** piece\_of\_pie(n, nums, bob, alice):

bob[**-**1] **=** nums[**-**1] *# Последнему элементу массива bob присвоить последний элемент массива nums*

alice[**-**1] **=** nums[**-**1] *# Последнему элементу массива alice присвоить последний элемент массива nums*

**for** i **in** range(n **-** 2, **-**1, **-**1): *# Пролистываем массив в обратном порядке*

alice[i] **=** alice[i **+** 1] **+** nums[i]

bob[i] **=** max(bob[i **+** 1], alice[i] **-** bob[i **+** 1])

fin\_alice **=** alice[0] **-** bob[0]

fin\_bob **=** bob[0]

**return** f'Суммарные размеры кусочков пирога Алисы: {fin\_alice}, Боба: {fin\_bob}'

print('-----------------------------------------\n', 'Все данные вводить строго через пробел!',

'\n-----------------------------------------\n')

n **=** int(input('Введите количество кусочков пирога: '))

**if** n **not** **in** range(1, 51):

sys**.**exit(f'Число n = {n} не находится в диапазоне 1 <= X <= 50')

nums **=** list(map(int, input('Введите размеры кусочков: ')**.**split()))

**if** len(nums) **==** n:

**for** el **in** nums:

**if** el **not** **in** range(1, 10**\*\***5 **+** 1):

sys**.**exit(f'Число {el} не находится в диапазоне 1 <= X <= 100 000')

**else**:

sys**.**exit('Количество введеных размеров не совпадает с количеством кусочков пирога n!')

bob **=** [0] **\*** n *# Массив нулей*

alice **=** [0] **\*** n *# Массив нулей*

print('')

print(piece\_of\_pie(n, nums, bob, alice))

**Задача №4**

Мальчик Лёша любит слушать музыку со своего смартфона. Но памяти у смартфона не очень много, поэтому Лёша слушает любимые композиции в известной социальной сети НаСвязи.

К сожалению, интернет в Екатеринозаводске не очень быстрый, и песня загружается очень медленно. Но Лёша весьма нетерпелив. Песня длится ровно T секунд. Лёша прогружает первые S секунд композиции и включает песню. Когда воспроизведение доходит до момента, который ещё не прогрузился, Лёша моментально включает песню заново (при этом загруженная часть песни остаётся в памяти телефона, и процесс скачивания продолжается с того же места), и так происходит до тех пор, когда песня скачается полностью и Лёша дослушает до конца. За q секунд реального времени интернет позволяет скачать q - 1 секунду трека.

Подскажите Лёше, сколько раз он запустит песню, включая самый первый запуск.

Входные данные

В единственной строчке записаны три целых числа T, S, q (2 ≤ q ≤ 104, 1 ≤ S < T ≤ 105).

Выходные данные

Выведите одно целое число — количество запусков композиции.

**Решение**

**import** sys

**def** music(t, s, q):

count **=** 1

s **=** s **\*** q *# Загружено изначально*

**while** t **>** s: *# Пока длительность больше длительности загруженной части музыки*

s **=** s **\*** q *# Считаем, сколько загрузилось в момент прослушивания*

count **+=** 1

**return** f'Количество запусков композиции составляет: {count}'

print('-----------------------------------------\n', 'Все данные вводить строго через пробел!',

'\n-----------------------------------------\n')

t, s, q **=** list(map(int, input('Введите значения параметров T, S, q: ')**.**split()))

**if** t **not** **in** range(s, 106):

sys**.**exit(f'Число T = {t} не находится в диапазоне 1 <= S < T <= 105')

**if** s **not** **in** range(1, t):

sys**.**exit(f'Число S = {s} не находится в диапазоне 1 <= S < T <= 105')

**if** q **not** **in** range(2, 105):

sys**.**exit(f'Число q = {q} не находится в диапазоне 2 <= q <= 104')

print('')

print(music(t, s, q))

Задача №5

Спортсмен Вася участвует в соревнованиях по спортивному ориентированию. На соревновательной прямой расположены n контрольных пунктов с координатами x1, x2, ..., xn. Вася стартует из точки с координатой a. Ему необходимо посетить как минимум n - 1 контрольный пункт, чтобы финишировать. Участники могут посещать контрольные пункты в произвольном порядке.

Васю интересует, какие контрольные пункты и в каком порядке ему нужно посещать, чтобы пройденное им расстояние было минимально возможным. Помогите ему вычислить это расстояние.

Входные данные

В первой строке входных данных даны два целых числа n и a (1 ≤ n ≤ 100 000,  - 1 000 000 ≤ a ≤ 1 000 000) — количество контрольных пунктов и начальная координата Васи соответственно.

Во второй строке записаны n целых чисел x1, x2, ..., xn ( - 1 000 000 ≤ xi ≤ 1 000 000) — координаты контрольных пунктов.

Выходные данные

Выведите одно целое число — минимальное суммарное расстояние, которое нужно пройти Васе, чтобы посетить как минимум n - 1 контрольный пункт.

**Решение**

**import** sys

**def** check\_point(n, a, x):

**if** n **==** 1:

**return** 0

**elif** n **<=** 2: *# Если два КП, то смотрим до какого ближе, где минимальный путь*

**return** min(abs(x[0] **-** a), abs(x[n**-**1] **-** a))

**else**: *# Аналогично, если более 2 КП, то ищем минимум*

temp **=** x[n**-**2] **-** x[0] **+** min(abs(x[0] **-** a), abs(x[n**-**2] **-** a))

**return** min(temp, x[n**-**1] **-** x[1] **+** min(abs(x[1] **-** a), abs(x[n**-**1] **-** a)))

print('-----------------------------------------\n', 'Все данные вводить строго через пробел!',

'\n-----------------------------------------\n')

n, a **=** list(map(int, input('Введите количество контрольных пунктов и начальную координату: ')**.**split()))

**if** n **not** **in** range(1, 10**\*\***5 **+** 1):

sys**.**exit(f'Число n = {n} не находится в диапазоне 1 <= n <= 100 000')

**if** a **not** **in** range(**-**100**\*\***3, 100**\*\***3 **+** 1):

sys**.**exit(f'Число a = {a} не находится в диапазоне -1 000 000 <= a <= 1 000 000')

x **=** list(map(int, input('Введите координаты контрольных пунктов: ')**.**split()))

**if** len(x) **==** n:

**for** el **in** x:

**if** el **not** **in** range(**-**100**\*\***3, 100**\*\***3 **+** 1):

sys**.**exit(f'Число {el} не находится в диапазоне 1 <= X <= 100 000')

**else**:

sys**.**exit('Количество введеных координат не совпадает с количеством контрольных пунктов n!')

print('')

print('Минимальное суммарное расстояние составляет:', check\_point(n, a, x))

Задача №6

Иннокентий очень любит чай и сегодня хочет выпить ровно n чашек чая. Он бы выпил и больше, но у него осталось ровно n пакетиков чая, a из которых — зеленый чай и b — черный.

Иннокентий не любит пить более, чем k раз подряд один и тот же чай (зеленый или черный). Перед вами стоит задача определить порядок заваривания пакетиков, при котором Иннокентий сможет выпить n чашек чая, при этом он не будет пить один и тот же чай более k раз подряд, либо сообщить, что это невозможно. Каждый пакетик должен быть заварен ровно один раз.

Входные данные

В первой строке следуют четыре целых числа n, k, a и b (1 ≤ k ≤ n ≤ 105, 0 ≤ a, b ≤ n) — количество чашек чая, которые хочет выпить Иннокентий, максимальное количество чашек одного и того же чая, которые Иннокентий может выпить подряд, количество пакетиков с зеленым чаем и количество пакетиков с черным чаем. Гарантируется, что a + b = n.

Выходные данные

Если выпить n чашек чая не удастся, выведите «NO» (без кавычек).

В противном случае, выведите строку длины n, состоящую из символов 'G' и 'B'. Если очередной символ равен 'G', то очередная кружка должна быть с зеленым чаем. Если очередной символ равен 'B', то очередная кружка должна быть с черным чаем. Если ответов несколько, разрешается вывести любой из них.

**Решение**

**import** sys

**def** green\_black\_tea(n, k, a, b):

temp **=** abs(a **-** b) *# Смотрим, какого чая больше*

check **=** max(1, k **-** 1)

ans **=** ''

**if** a **==** b: *# Если поровну, то чередуем зеленый и черный*

ans **=** 'GB' **\*** ((n **+** 1) **//** 2)

**elif** a **>** b:

ans **+=** ('G' **\*** k **+** 'B') **\*** (temp **//** check)

ans **+=** 'G' **\*** max(1, (temp **%** check)) **+** 'B'

ans **+=** 'GB' **\*** (n **//** 2)

**else**:

ans **+=** ('B' **\*** k **+** 'G') **\*** (temp **//** check)

ans **+=** 'B' **\*** max(1, (temp **%** check)) **+** 'G'

ans **+=** 'BG' **\*** (n **//** 2)

ans **=** ans[:n]

**if** ans**.**count('G') **==** a **and** ans**.**count('B') **==** b :

**return** ans

**else**:

**return** 'NO'

print('-----------------------------------------\n', 'Все данные вводить строго через пробел!',

'\n-----------------------------------------\n')

n, k, a, b **=** list(map(int, input('Введите значение параметров n, k, a, b: ')**.**split()))

**if** n **not** **in** range(k, 106):

sys**.**exit(f'Число {n} не находится в диапазоне 1 <= k <= n <= 105')

**if** k **not** **in** range(1, n **+** 1):

sys**.**exit(f'Число {k} не находится в диапазоне 1 <= k <= n <= 105')

**if** a **<** 0:

sys**.**exit(f'Число {a} должно быть больше или равно нулю!')

**if** b **>** n:

sys**.**exit(f'Число {b} должно быть меньше или равно n!')

**if** a **+** b **!=** n:

sys**.**exit('Количество зеленого (а) и черного (b) чая меньше, чем n! \n\t\t\t\t\t a + b = n')

print('')

print('Полученная комбинация:', green\_black\_tea(n, k, a, b))

Задача №7

Вы — искатель приключений. Сейчас Вы путешествуете по злому храму. Одолев пару злых зомби, Вы прибыли в квадратную комнату, состоящую из плиток, выложенных в таблицу размера n × n. Строки пронумерованы от 1 до n сверху вниз, а столбцы пронумерованы с 1 до n слева направо. В дальнем углу комнаты находится дверь, закрытая злыми магическими силами. На двери написано следующее:

Очищение от всякого зла пробудит дверь! Так как Вы довольно опытный искатель приключений, Вы мгновенно понимаете, в чем тут дело. Вы замечаете, что каждая ячейка в таблице изначально является злой. Надо очистить все эти ячейки.

Единственный известный Вам способ очищения плиток — заклинание «Очищение». Заклинание накладывается на одну плитку, а затем все ячейки в той же строке, а также все ячейки в том же столбце (включая выбранную плитку) очищаются! Разрешается очищать клетку несколько раз.

Вы хотели бы очистить все n × n ячеек, накладывая заклинание очищения как можно меньше раз. В теории все просто, но вы только что заметили, что есть еще особо злые плитки. На них заклинание очищения нельзя накладывать, даже после того, как они были очищены. Но эти плитки все же можно очистить, если наложить заклинание очищения на ячейку в той же строке или столбце.

Пожалуйста, найдите способ очистить все ячейки минимальным количеством накладываемых заклинаний. Если такого способа нет, выведите -1.

Входные данные

В первой строке записано единственное целое число n (1 ≤ n ≤ 100). В следующих n строках записано по n символов: j-ый символ i-ой строки обозначает ячейку, расположенную в строке i и столбце j. Особо злые плитки заданы символом «E», остальные — символом «.».

Выходные данные

Если очистить все клетки не удастся, то выведите -1. В противном случае, если Ваше решение накладывает x заклинаний очищения, выведите x строк. Каждая строка должна состоять из двух целых чисел, обозначающих номер строки и столбца ячейки, на которую следует наложить заклинание очищения.

**Решение**

**import** sys

**def** wizard(n, coord):

result **=** ''

*# zip() - объединяет в кортежи элементы из последовательностей переданных в качестве аргументов*

**if** 'E' **\*** n **in** coord **and** ('E',) **\*** n **in** zip(**\***coord): *# Если столбец и строчка ПОЛНОСТЬЮ из 'Е'*

**return** **-**1

**else**:

**for** j **in** range(n):

**for** i **in** range(n):

**if** coord[i][j] **==** '.':

result **+=** str(i **+** 1) **+** ' ' **+** str(j **+** 1) **+** '\n'

**break**

**return** result

print('-----------------------------------------\n', 'Все данные вводить строго через пробел!',

'\n-----------------------------------------\n')

n **=** int(input('Введите размер комнаты машстабом n×n: '))

**if** n **not** **in** range(1, 101):

sys**.**exit(f'Число {n} не находится в диапазоне 1 <= n <= 100')

print('Введите "характер" плитки:\n\tЕ - "злая" плитка\n\t. - "хорошая" плитка\n')

coord **=** [input()**.**strip() **for** \_ **in** range(n)]

print()

print('Координаты для лучшей очистки от нечести:\n')

print(wizard(n, coord))

Задача №8

Принцесса собирается сбежать из пещеры дракона. Это нужно как следует спланировать.

Принцесса бегает со скоростью vp миль в час, дракон летает со скоростью vd миль в час. Дракон обнаружит побег через t часов и немедленно бросится в погоню. Затея кажется безнадежной, но принцесса заметила, что дракон очень жаден и не очень умен. Чтобы задержать его, принцесса решает прихватить с собой несколько драгоценностей из его сокровищницы. Когда дракон догоняет принцессу, она может бросить на землю драгоценность; тогда дракон остановится, подберет упавшее, вернется в пещеру и потратит f часов в самой пещере на возвращение вещицы на место и наведение порядка в сокровищнице, после чего вновь отправится в погоню, начиная все с самого начала.

Предполагая, что принцесса будет бежать по прямой без остановок, сколько драгоценностей ей нужно взять с собой, чтобы успеть добраться до королевского замка, расположенного на расстоянии c миль от пещеры дракона? Если дракон догоняет принцессу в тот момент, когда она добегает до замка, считается, что она успела раньше (дополнительная драгоценность не нужна).

Входные данные

Входные данные содержат целые числа vp, vd, t, f и c (1 ≤ vp, vd ≤ 100, 1 ≤ t, f ≤ 10, 1 ≤ c ≤ 1000), каждое число записано в отдельной строке.

Выходные данные

Выведите минимальное количество драгоценностей, необходимых для того, чтобы побег удался.

**Решение**

**import** sys

**def** princess(vp, vd, t, f, c):

count **=** 0

**while** vp **<** vd:

temp **=** (vp **\*** t) **/** (vd **-** vp) *# 1 скобка - куда принцесса успеет добежать, 2 - разница*

**if** temp **\*** vd **>=** c: *# Если принцесса успела добежать до замка, то выход из цикла*

**break**

t **+=** 2 **\*** temp **+** f *# Время, которое тратит дракон после выбрасывания драгоценностей*

count **+=** 1

**return** f'Минимальное количество драгоценностей составляет: {count}'

print('-----------------------------------------\n', 'Все данные вводить строго через пробел!',

'\n-----------------------------------------\n')

vp **=** int(input('Введите скорость бега принцессы: '))

**if** vp **<** 1:

sys**.**exit(f'Число vp = {vp} меньше, чем 1!')

vd **=** int(input('Введите скорость полета дракона: '))

**if** vd **>** 100:

sys**.**exit(f'Число vd = {vd} больше, чем 100!')

t **=** int(input('Введите количество часов, через которое дракон обнаружит пропажу: '))

**if** t **<** 1:

sys**.**exit(f'Число t = {t} меньше, чем 1!')

f **=** int(input('Введите количество часов на возвращение вещицы: '))

**if** f **>** 10:

sys**.**exit(f'Число f = {f} больше, чем 10!')

c **=** int(input('Введите расстояние замка принцессы до пещеры дракона: '))

**if** c **not** **in** range(1, 1001):

sys**.**exit(f'Число c = {c} не находится в диапазоне 1 <= c <= 1000')

print()

print(princess(vp, vd, t, f, c))

Задача №9

Маленький Слоник любит играть с цветными карточками.

У него есть n карточек, каждая имеет ровно два цвета (цвет на передней стороне и цвет на задней стороне). Изначально все карточки лежат на столе передней стороной вверх. За один шаг Маленький Слоник может перевернуть на другую сторону любую карточку. Маленький Слоник считает набор карточек на столе веселым, если хотя бы половина карточек имеет одинаковый цвет (для каждой карточки рассматривается цвет стороны, которой она лежит вверх).

Помогите Маленькому Слонику найти минимальное количество шагов необходимых для превращения набора из n карточек в веселый.

Входные данные

В первой строке задано единственное целое число n (1 ≤ n ≤ 105) — количество карточек. Следующие n строк содержат описание всех карточек, по одной карточке на строку. Карточки описываются парой целых положительных чисел, не превосходящих 109, — цветами на обеих сторонах. Первое число в строке — это цвет на передней стороне карточки, второе — на задней. Цвет на передней стороне может совпадать с цветом на задней.

Числа в строках разделяются единичными пробелами.

Выходные данные

В единственной строке выведите единственное целое число — искомое минимальное количество шагов. Если превратить набор в веселый невозможно, выведите -1.

**Решение**

**import** sys

**def** cards(n):

d **=** {}

**for** i **in** range(n):

a, b **=** map(int, input(f'Введите передний и оборотный цвета карточки №{i **+** 1}: ')**.**split())

**if** a **>** 109:

sys**.**exit(f'Число a = {a} больше, чем 109!')

**elif** b **>** 109:

sys**.**exit(f'Число b = {b} больше, чем 109!')

**if** a **not** **in** d:

d[a] **=** [0, 0]

d[a][0] **+=** 1

**if** a **!=** b:

**if** b **not** **in** d:

d[b] **=** [0, 0]

d[b][1] **+=** 1

half **=** (n **+** 1) **//** 2

result **=** 2 **\*** n

**for** k, v **in** d**.**items():

**if** sum(v) **>=** half:

result **=** min(result, half **-** v[0])

**if** result **==** 2 **\*** n:

**return** **-**1

**else**:

**return** max(0, result)

print('-----------------------------------------\n', 'Все данные вводить строго через пробел!',

'\n-----------------------------------------\n')

n **=** int(input('Введите количество карточек: '))

**if** n **not** **in** range(1, 106):

sys**.**exit(f'Число n = {n} не находится в диапазоне 1 <= n <= 105')

print()

print('Минимальное количество шагов составляет: ', cards(n))

Задача №10

Вася живет посередине Программистской ветки метро. У него есть две девушки: Даша и Маша, живущие на разных концах ветки, каждая не ведающая о существовании другой.

Когда у Васи появляется свободное время, он едет к одной из своих девушек. Он спускается в метро в некоторый момент времени, ждет первый поезд и едет на нем до конца ветки к соответствующей девушке. Однако поезда ходят с разной частотой: в Дашином направлении станцию проезжает один поезд в a минут, а в Машином — один поезд в b минут. В случае, если поезда подошли одновременно, Вася едет в направлении с меньшей частотой хождения поездов, то есть к той девушке, в чью сторону поезда ходят реже (см. пояснение к третьему примеру).

Известно, что поезда начинают свое хождение одновременно до появления Васи. То есть расписание поездов таково, что существует момент времени, когда два поезда приезжают одновременно.

Помогите Васе посчитать, к какой девушке он будет попадать чаще.

Входные данные

В первой строке записано два целых числа a и b (a ≠ b, 1 ≤ a, b ≤ 106).

Выходные данные

Выведите «Dasha», если Вася будет чаще попадать к Даше, «Masha», если к Маше, и «Equal», если одинаково часто к обеим девушкам.

**Решение**

**import** sys

**from** math **import** **\***

**def** lovelas(a, b):

**if** gcd(a, b) **==** abs(b **-** a): *# gcd - Функция для вычисления НОД*

**return** ('Equal')

**elif** (a **<** b):

**return** ('Dasha')

**else**:

**return** ('Masha')

print('-----------------------------------------\n', 'Все данные вводить строго через пробел!',

'\n-----------------------------------------\n')

a, b **=** list(map(int, input('Введите частоту прибытия для поездов a и b: ')**.**split()))

**if** a **!=** b:

**if** a **<** 1:

sys**.**exit(f'Число a = {a} меньше, чем 1!\nВведите такое число, что a >= 1')

**if** b **>** 106:

sys**.**exit(f'Число b = {b} больше, чем 106!\nВведите такое число, что b <= 106')

**else**:

sys**.**exit('Частота прибытия поездов одинаковая!\nВведите такие числа, что a != b')

print()

print('Полученный результат:', lovelas(a, b))

Вывод

В ходе выполнения курсовой работы я успешно выполнил все поставленные задачи и закрепил полученные в ходе изучения данной дисциплины знания.