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

**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

“Системы и алгоритмы обработки данных”

Выполнил студент группы

БФИ2001 Никитин Г.А

Вариант №16

Научный руководитель: Симонов С.Е

Москва 2021

Оглавление

Введение3

## Задачи курсовой работы и их решения3

Заключение51

Приложение 54

## Введение

## Для выполнения курсовой работы необходимо найти решение нескольких задач в соответствии с вариантом.

## Каждая задача имеет стоимость в 2 балла. Итоговая оценка выставляется в соответствии с таблицей:

|  |  |
| --- | --- |
| Кол-во баллов | оценка |
| 0-10 | Неудовлетворительно |
| 12-14 | Удовлетворительно |
| 16-18 | Хорошо |
| 20 | Отлично |

## Задачи курсовой работы и их решения

**Задача 1**

Финалистами соревнований «Russian Code Cup» в 2214 году будут участники, ставшие победителями в одном из отборочных раундов.

Отборочные раунды делятся на основные и дополнительные. Каждый из основных отборочных раундов должен состоять из c задач, а победителями раунда считаются n человек, занявшие первые места в этом раунде. Каждый из дополнительных отборочных раундов состоит из d задач. Победителем дополнительного раунда становится один человек. Кроме этого, на финал без конкурса приглашаются k победителей финалов прошлых лет.

В результате всех отборочных раундов в финал должно пройти не менее n·m человек. Каким образом нужно организовать отборочные раунды, чтобы в результате всех отборочных раундов в финал прошли не менее n·m человек, а при этом суммарное количество использованных в раундах задач было как можно меньше?

Входные данные Первая строка содержит два целых числа c и d (1 ≤ c, d ≤ 100) — количество задач в основном и дополнительном раундах соответственно. Вторая строка содержит два целых числа n и m (1 ≤ n, m ≤ 100). Наконец, третья строка содержит целое число k (1 ≤ k ≤ 100) — число заранее отобранных победителей.

Выходные данные В первой строке выведите единственное целое число — минимальное количество задач, которое нужно подготовить членам жюри.

Примеры  
входные данные  
1 10  
7 2  
1  
выходные данные  
2

Решение задачи:

# task1

# s = input("Введите c и d через пробел:")

s1 = '1 10'

# s = input("Введите n и m через пробел:")

s2 = '7 2'

# s = input("Введите k:")

s3 = '1 3'

def task1 (s1, s2, s3):

s1 = s1.split(" ")

if(len(s1) > 2):

return 'ошибка данных'

exit(0)

c = int(s1.pop(0))

d = int(s1.pop(0))

if not (1 <= c <= 100) or not (1 <= d <= 100):

return 'ошибка данных'

exit(0)

# s = input("Введите n и m через пробел:")

s2 = s2.split(" ")

if (len(s2) > 2):

return 'ошибка данных'

exit(0)

n = int(s2.pop(0))

m = int(s2.pop(0))

if not (1 <= n <= 100) or not (1 <= m <= 100):

return 'ошибка данных'

exit(0)

# s = input("Введите k:")

s3 = s3.split(" ")

if (len(s3) > 1):

return 'ошибка данных'

exit(0)

k = int(s3.pop(0))

if not (1 <= k <= 100):

return 'ошибка данных'

exit(0)

total\_winners = n \* m # всего финалистов

# требуемое количество победтелей

need\_winners = total\_winners - k

# если за один,основной, раунд соревнований нельзя закрыть все требуемые позиции - 1(победитель дополнительного раунда), то необходимо провести еще один основной раунд

# если проводится дополнительный раунд соревнований, то приоритет у победителей основного раунда

main\_round\_count = 0 # количество основных раундов

while int((need\_winners) - (main\_round\_count \* n)) >= main\_round\_count:

main\_round\_count = main\_round\_count + 1

task\_count = 0

if need\_winners - main\_round\_count \* n <= main\_round\_count:

task\_count = c \* main\_round\_count

return task\_count

else:

task\_count = c \* main\_round\_count + d \* (need\_winners - main\_round\_count \* n)

return task\_count

print(task1(s1, s2, s3))



Рисунок 1- результат работы программы

**Задача 2**

Ярослав, Андрей и Роман могут часами напролет играть в кубики. Но это игра для трех человек, поэтому, когда Роман отсутствует, Ярослав и Андрей играют в другую игру.

Роман оставляет каждому из них по слову, состоящему из 2·n символов «0» или «1». После этого игроки начинают ходить по очереди. Ярослав ходит первым. На своем ходе игрок должен выбрать целое число от 1 до 2·n, которое не выбиралось никем из игроков ранее, и выписать себе на листочек соответствующий символ из своей строки.

Обозначим слово, которое оставили Ярославу, как s = s1s2... s2n. Аналогично, обозначим слово, которое оставили Андрею, как t = t1t2... t2n. Тогда, если Ярослав выбрал на своем ходе число k, то он выпишет на свой листочек символ sk. Аналогично, если Андрей выбрал на своем ходе число r, то он выпишет на свой листочек символ tr.

Игра заканчивается, когда ни один из игроков не может сделать ход. После окончания игры, Ярослав составляет из символов, записанных на своем листочке, некоторое целое число (символы можно переставлять в любом порядке), тоже самое делает Андрей (полученные числа могут содержать лидирующие нули). Побеждает тот, у кого получилось большее число. Если получившиеся числа равны, то объявляется ничья.

Вам заданы две строки s и t. Определите исход игры при оптимальной игре Ярослава и Андрея.

Входные данные В первой строке записано целое число n (1 ≤ n ≤ 106). Во второй строке записана строка s — слово Ярослава. В третьей строке записана строка t — слово Андрея.

Гарантируется, что оба слова состоят из 2·n символов «0» и «1».

Выходные данные Выведите «First», если при оптимальной игре побеждает Ярослав, «Second» — в случае победы Андрея и «Draw», если при оптимальной игре будет ничья. Слова выводите без кавычек.

входные данные  
3  
110110  
001001  
выходные данные  
First  
входные данные  
3  
111000  
000111  
выходные данные  
Draw

Решение задачи:

#task2  
  
n = 3  
s1 = '110000'  
s2 = '000111'  
  
# проверить слово на валидность  
def check\_word(s):  
 for i in range(0, len(s)):  
 if s[i] != '1' and s[i] != '0': return 0  
 return 1  
  
# конвертировать строку в список  
def convert\_string\_to\_list(str):  
 list = []  
 for i in range(0, len(str)):  
 list.append(str[i])  
 return list  
  
# определить победителя  
def calculate\_result(list1, list2):  
 if list1.count('1') > list2.count('1'):  
 return "First"  
 elif list1.count('1') < list2.count('1'):  
 return "Second"  
 else:  
 return "Draw"  
  
if not (1 <= n <= 106) or int(n) \* 2 != len(s1) or int(n) \* 2 != len(s2) or not check\_word(s1) or not check\_word(s2):  
 print("Ошибка данных")  
 exit(0)  
s1 = convert\_string\_to\_list(s1)  
s2 = convert\_string\_to\_list(s2)  
l1 = [] # ходы Ярослава  
l2 = [] # ходы Андрея  
  
# чтобы победить каждый из игроков должен набоать максимально возможное количество "1"  
for i in range(0, n):  
 # ход Ярослава  
 if s1.count('1') > 0:  
 l1.append(s1.pop(s1.index('1')))  
 else:  
 l1.append(s1.pop())  
 # ход Андрея  
 if s2.count('1') > 0:  
 l2.append(s2.pop(s2.index('1')))  
 else:  
 l2.append(s2.pop())  
  
print(calculate\_result(l1, l2))



Рисунок 2- результат работы программы

**Задача 3**

Счастливый PMP учится в ВУЗе на первом курсе, где он изучает алгоритмические задачи. PMP обожает алгоритмические игры.

Один студент постарше дал счастливому PMP занятную игру. PMP даны две перестановки чисел от 1 до n, от него требуется преобразовать первую перестановку во вторую. За один шаг можно убрать из перестановки чисел последнее число и вставить его обратно в произвольное место. Другими словами, можно либо вставить последнее число между любыми двумя числами, идущими одно за другим, либо вставить число в начало перестановки.

Счастливый PMP знает алгоритм, решающий эту задачу, но он слишком медленный. PMP хочет знать минимальное количество шагов, за которое он может преобразовать первую перестановку во вторую.

Входные данные В первой строке записано единственное целое число n (1 ≤ n ≤ 2·105) количество чисел в каждой из двух заданных перестановок.

В следующей строке записано n целых чисел, разделенных пробелом — первая перестановка. Каждое число от 1 до n встретится в перестановке ровно один раз.

Следующая строка описывает вторую перестановку в аналогичном формате.

Выходные данные Выведите единственное целое число, обозначающее минимальное количество шагов, необходимых для преобразования первой перестановки во вторую.

Примеры  
входные данные  
3  
3 2 1  
1 2 3  
выходные данные  
2  
входные данные  
5  
1 2 3 4 5  
1 5 2 3 4  
выходные данные  
1

Решение задачи:

n = 5  
s1 = "1 2 3 4 5"  
s2 = "1 5 2 3 4 "  
  
# конвертировать строку в список  
def convert\_string\_to\_list(str):  
 list = []  
 str = str.split(" ")  
 for i in range(0, len(str)):  
 list.append(int(str[i]))  
 return list  
  
  
# проверить исходные данные  
def validate\_data(list1, list2, n):  
 if not (1 <= n <= 2 \* 105) or len(list1) != len(list2) or len(list1) != n:  
 return False  
 l1 = list1.copy()  
 l2 = list2.copy()  
 l1.sort()  
 l2.sort()  
 # отсортированные списки должны быть идентиичны  
 if l1 != l2:  
 return False  
  
 # списки должны быть от 1 до n  
 for i in range(1, n):  
 if l1[i] != l1[i - 1] + 1:  
 return False  
  
 return True  
  
# функция ищет кратчайщее пребразование из l1 в l2  
# иначе начинаем с начала списка  
# ищем 2 идущих по порядку числа и вставляем последнее число списка между ними  
# далее рекурсивно вызываем функцию  
# так проделываем для всех пар чисел списка на первом уровне  
def find\_answer(l1, l2, layer):  
 # делаем искуственное ограничение на глубину рекурсии равную 2 \* n  
 # то есть, если решение не найдено за 2n шагов, то дальще искать не имеет смысла  
 if layer > len(l1) \* 2:  
 return len(l1) \* 2  
 # если l1 и l2 равны возвращаем layer - уровень рекурсии(колличество сделанных операций)  
 # if compare\_list(l1, l2):  
 if l1 == l2:  
 return layer  
 # вставляем последнее число перед первым  
 ll = l1.copy()  
 ll.insert(0, ll.pop(len(ll)-1))  
 # и передаем в рекурсивно в саму себя  
 min\_layer = find\_answer(ll, l2, layer + 1)  
 # последовательно вставляем последний элемент списка  
 # между всеми парами идущими по порядку  
 for i in range(1, len(l2)):  
 ll = l1.copy()  
 ll.insert(i, ll.pop(len(ll)-1))  
 min\_layer = min(min\_layer, find\_answer(ll, l2, layer + 1))  
 return min\_layer  
  
  
# конвертируем исходные данные в список  
start\_list = convert\_string\_to\_list(s1)  
destination\_list = convert\_string\_to\_list(s2)  
  
# проверяем ИД на соответствие условиям задачи  
if not validate\_data(start\_list, destination\_list, n):  
 print("Исходные данные не отвечают условиям задачи")  
 exit(0)  
  
print(find\_answer(start\_list, destination\_list, 0))



Рисунок 3- результат работы программы

**Задача 4**

Мальчик Петя обожает шахматы. Он даже придумал свою собственную шахматную фигуру — полуконя. Полуконь может ходить в любом из следующих четырех направлений: на 2 клетки вперед и 2 вправо, на 2 клетки вперед и 2 влево, на 2 клетки назад и 2 вправо или на 2 клетки назад и 2 влево. Разумеется, полуконь не может выходить за пределы шахматной доски.

На стандартной шахматной доске Петя поставил двух полуконей. Петя одновременно делает ходы обоими полуконями. Клетки доски достаточно большие, поэтому полукони после некоторого хода могут встретиться, то есть оказаться в одной и той же клетке. После встречи полукони могут продолжить ходить, поэтому возможно, что в дальнейшем они вновь встретятся. Пете интересно, существует ли такая последовательность ходов, при которой полукони встретятся. Некоторые клетки Петя посчитал плохими, то есть не подходящими для встречи. Полукони могут ходить по этим клеткам, но встречи в этих клетках не засчитываются.

Петя подготовил несколько шахматных досок. Помогите Пете узнать для каждой доски, могут ли полукони встретиться на некоторой подходящей для этого клетке.

Пожалуйста, обратите внимание на разбор тестового примера.

Входные данные В первой строке задано число t (1 ≤ t ≤ 50) — количество досок. Каждая доска описывается матрицей символов, состоящей из 8 строк и 8 столбцов. Матрица состоит из символов «.», «#», «K», обозначающих соответственно пустую клетку, плохую клетку и клетку, в которой стоит полуконь. Гарантируется, что на доске стоят ровно 2 полуконя. Клетки, занимаемые полуконями, считаются подходящими для встречи. Тесты разделены пустой строкой.

Выходные данные Выведите для каждого теста в отдельной строке ответ на задачу: «YES», если полукони смогут встретиться, и «NO» в противном случае.

Примеры  
входные данные  
2  
........  
........  
......#.  
K..##..#  
.......#  
...##..#  
......#.  
K.......  
  
........  
........  
..#.....  
..#..#..  
..####..  
...##...  
........  
....K#K#  
выходные данные  
YES  
NO  
  
Примечание  
Рассмотрим первую доску из примера. Будем считать, что строки и столбцы матрицы нумеруются от 1 до 8 сверху вниз и слева направо соответственно. Полукони могут встретиться, например, в клетке (2, 7). Полуконь из клетки (4, 1) переходит в клетку (2, 3), а полуконь из клетки (8, 1) — в клетку (6, 3). Далее оба полуконя переходят в (4, 5), но эта клетка — плохая, поэтому они вместе переходят в клетку (2, 7).  
На второй доске полукони никогда не встретятся.

**Задача 5**

Маленький пингвин Поло любит свою родную деревню. В этой деревне n домов, пронумерованных целыми числами от 1 до n. На каждом доме висит табличка с целым числом, на i-том доме висит табличка с целым числом pi (1 ≤ pi ≤ n).

Маленький пингвин Поло очень любит гулять по этой деревне. Прогулка выглядит следующим образом. Сначала он стоит около дома с номером x. Потом пингвин идет к дому, номер которого написан на табличке дома x (то есть к дому px), затем к дому, номер которого написан на табличке дома px (то есть к дому ppx), и так далее.

Известно, что:

Начав прогулку от любого дома с номером от 1 до k, включительно, он может дойти до дома с номером 1. Начав прогулку от любого дома с номером от k + 1 до n, включительно, он точно не может дойти до дома с номером 1. Начав прогулку от дома с номером 1, пингвин Поло может попасть обратно к дому с номером 1 через некоторое ненулевое количество переходов от дома к дому.

Вам нужно найти, сколькими способами можно написать числа на табличках домов, чтобы описанные выше три условия выполнялись. Выведите остаток от деления этого количества на 1000000007 (109 + 7).

Входные данные В единственной строке через пробел заданы два целых числа n и k (1 ≤ n ≤ 1000, 1 ≤ k ≤ min(8, n)) — количество домов и число k из условия.

Выходные данные В единственной строке выведите целое число — ответ на задачу по модулю 1000000007 (109 + 7).

Примеры  
входные данные  
5 2  
выходные данные  
54

**Задача 6**

Facetook — широко известная социальная сеть. Недавно в ней появилась новая функция — стена с приоритетами. Эта функция позволяет сортировать всех ваших друзей по коэффициенту дружбы. Чем больше вы общаетесь с другом, тем больше будет его коэффициент дружбы.

На коэффициент дружбы влияет 3 типа действий:

1. человек X написал на стене человек Y ("X posted on Y's wall") — 15 очков;
2. человек X прокомментировал сообщение человека Y ("X commented on Y's post") — 10 очков;
3. человеку X понравилось сообщение человека Y ("X likes Y's post") — 5 очков. X и Y — два различных имени. Каждое действие увеличивает коэффициент дружбы между X и Y (и наоборот) на данное количество очков (коэффициент дружбы между X и Y равен коэффициенту дружбы между Y и X).

Вам дано n действий в формате, описанном выше. Выведите все различные имена, которые встречаются в списке действий, в отсортированном порядке в соответствие с коэффициентом дружбы с вами.

Входные данные В первой строке записано ваше имя. Во второй строке записано целое число n, количество действий (1 ≤ n ≤ 100). Далее следует n строк. Гарантируется, что каждая строка содержит описание ровно одного действия в описанном выше формате. Между каждой парой слов в строке есть ровно один пробел, никаких других пробелов в строке нет. Все буквы имеют нижний регистр. Все имена во входных данных имеют длину от 1 до 10 строчных латинских букв.

Выходные данные Выведите m строк, где m — количество различных имен во входных данных (исключая вас). Каждая строка должна содержать ровно одно имя. Имена должны быть отсортированы в по убыванию коэффициента дружбы с вами (т. е. люди с большим коэффициентом дружбы должны идти раньше). Если два или больше человек имеют одинаковый коэффициент дружбы, выводите их в алфавитном (лексикографическом) порядке.

Учтите, что нужно выводить все имена, присутствующие во входных данных (исключая вас), даже если человек имеет нулевой коэффициент дружбы с вами.

Лексикографическое сравнение реализует оператор "<" в современных языках программирования. Строка a лексикографически меньше строки b, если либо a является префиксом b, либо существует такое i (1 ≤ i ≤ min(|a|, |b|)), что ai < bi, а для любого j (1 ≤ j < i) aj = bj, где |a| и |b| обозначают длины строк a и b соответственно.

Примеры  
входные данные  
ahmed  
3  
ahmed posted on fatma's wall  
fatma commented on ahmed's post  
mona likes ahmed's post  
выходные данные  
fatma  
mona

Решение задачи:

#task6  
  
in\_data = [  
 "ahmed",  
 3,  
 "ahmed posted on fatma's wall",  
 "fatma commented on ahmed's post",  
 "mona likes ahmed's post"]  
  
# функция сортировки списка  
# пузырьком  
def sort\_list(l):  
 for i in range(1, len(l)):  
 for j in range(i, len(l)):  
 if l[j][1] > l[j - 1][1] or (l[j][1] == l[j - 1][1] and l[j-1][0] > l[j][0]):  
 buf = l[j]  
 l[j] = l[j - 1]  
 l[j - 1] = buf  
 return l  
  
# ищем в списке имя и увеличиваем вес или создаем запись  
def append\_to\_list(name, weight, list):  
 fnd = 0  
 for i in range(0, len(list)): # ищем в списке результатов имя друга для увеличения его коэффициента  
 if list[i][0] == name:  
 list[i][1] = list[i][1] + weight  
 fnd = 1  
 break # нашли продолжать цикл нет необходимости  
 if not fnd: # если не найден, то добавляем  
 list.append([name, weight])  
 return list  
  
# функция проверяет входные данные  
# и формирует список коэффициентов дружбы по каждой уникальной персоне  
# входным параметром являются все исходные данные  
  
def make\_data(list):  
 score\_count = list[1] # количество оценок в переданном списке  
 if score\_count != len(list) - 2:  
 return False  
 my\_name = list[0] # мое имя  
 friend\_list = [] # список коэффициентов дружбы по всем оценкам  
  
 for i in range(2, len(list)):  
 s = list[i].split(" ") # разбираем строку с оценкой на составляющие  
 temp\_list = []  
 if s[1] == "posted" and s[2] == "on" and s[4] == "wall" and len(s) == 5:  
 name1 = s[0]  
 name2 = s[3].split("'")[0]  
 if name1 == my\_name or name2 == my\_name:  
 weight = 15  
 else:  
 weight = 0  
 elif s[1] == "commented" and s[2] == "on" and s[4] == "post" and len(s) == 5:  
 name1 = s[0]  
 name2 = s[3].split("'")[0]  
 if name1 == my\_name or name2 == my\_name:  
 weight = 10  
 else:  
 weight = 0  
 elif s[1] == "likes" and s[3] == "post" and len(s) == 4:  
 name1 = s[0]  
 name2 = s[2].split("'")[0]  
 if name1 == my\_name or name2 == my\_name:  
 weight = 5  
 else:  
 weight = 0  
 else:  
 return False  
 friend\_list.append([name1, name2, weight]) # сохраняем оценку  
  
 result\_list = [] # список коэффициентов дружбы  
 for i in range(0, len(friend\_list)):  
 if friend\_list[i][0] == my\_name:  
 result\_list = append\_to\_list(friend\_list[i][1], friend\_list[i][2], result\_list)  
 elif friend\_list[i][1] == my\_name:  
 result\_list = append\_to\_list(friend\_list[i][0], friend\_list[i][2], result\_list)  
 elif friend\_list[i][0] != my\_name and friend\_list[i][1] != my\_name and friend\_list[i][0] != friend\_list[i][1]:  
 result\_list = append\_to\_list(friend\_list[i][0], friend\_list[i][2], result\_list)  
 result\_list = append\_to\_list(friend\_list[i][1], friend\_list[i][2], result\_list)  
 else:  
 return False  
  
 result\_list = sort\_list(result\_list)  
 for\_return = []  
 # готовим к возврату только имена без весов отсортированные в порядке уменьшения коэффициента дружбы и алфавита  
 for i in range(0, len(result\_list)):  
 for\_return.append(result\_list.pop(0)[0])  
 return for\_return  
  
  
result = make\_data(in\_data)  
if not result:  
 print("Ошибка данных")  
print(result)

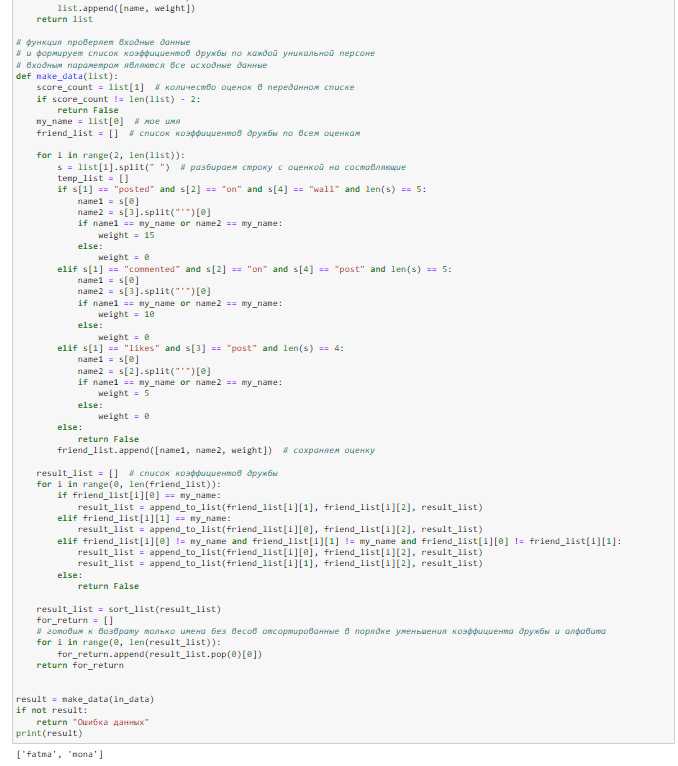


Рисунок 4- результат работы программы

**Задача 7**

Дальнейшие исследования мыслительных процессов зомби дали интересные результаты. Как мы знаем из предыдущей задачи, нервная система зомби состоит из n мозгов и m мозговых соединений, связывающих их воедино. Было замечено, что интеллектуальные способности зомби сильно зависят от топологии нервной системы. Точнее, определим как расстояние между двумя мозгами u и v (1 ≤ u, v ≤ n) как минимальное количество мозговых соединений, которые потребуется использовать, чтобы передать мысль между этими двумя мозгами. Мозговая задержка зомби определяется как максимальное расстояние между какой-нибудь парой мозгов. Ученые подозревают, что мозговая задержка является ключевым параметром определяющим, насколько умён данный зомби. Помогите им протестировать их гипотезу, написав программу для вычисления мозговой задержки данной нервной системы.

В данной задаче вы можете считать, что данная вам во входных данных мозговая система является правильной, то есть удовлетворяет пунктам (1) и (2) из лёгкой версии данной задачи.

Входные данные В первой строке входных данных записаны два целых числа n и m (1 ≤ n, m ≤ 100000), определяющих количество мозгов (пронумерованных от 1 до n) и количество мозговых связей в нервной системе, соответственно. В следующих m строках содержатся описания мозговых связей. Каждая связь задаётся как пара мозгов a b, которую она соединяет (1 ≤ a, b ≤ n, a ≠ b).

Выходные данные Выведите одно число — мозговую задержку данной нервной системы.

Примеры  
входные данные  
4 3  
1 2  
1 3  
1 4  
выходные данные  
2

Решение задачи:

# task7

# все связи между мозгами зомби это многосвязный граф

# depth\_search производит обход графа и ищет строит путь

def depth\_search(brain, zbrain\_link\_list, checked\_brain\_count, checked\_brain):

# Отметить как посещенные

checked\_brain[brain] = True

for i in range(0, len(zbrain\_link\_list[brain])):

# Если не посещали

if not checked\_brain[zbrain\_link\_list[brain][i]]:

depth\_search(zbrain\_link\_list[brain][i], zbrain\_link\_list, checked\_brain\_count, checked\_brain)

# Хранить максимум путей

checked\_brain\_count[brain] = max(checked\_brain\_count[brain], 1 + checked\_brain\_count[zbrain\_link\_list[brain][i]])

# Функция для добавления ребра

def add\_brain\_link(zbrain\_link\_list, brain1, brain2, brain\_count):

# проверка входных данных

if not (1 <= brain1 != brain2 and brain2 <= brain\_count) or not (1 <= brain\_count <= 10000):

return 'ошибка данных'

exit(0)

zbrain\_link\_list[brain1].append(brain2)

# Функция, которая возвращает самую длинную связь

def find\_longest\_path(zbrain\_link\_list, brain\_count):

# Dp array

checked\_brain\_count = [0] \* (brain\_count + 1)

# Проверочный массив, чтобы знать, если узел

# был посещен ранее или нет

checked\_brain = [False] \* (brain\_count + 1)

# Вызовите глубину для каждого не проверенного мозга

for i in range(1, brain\_count + 1):

if not checked\_brain[i]:

depth\_search(i, zbrain\_link\_list, checked\_brain\_count, checked\_brain)

path\_length = 0

# Пройти и найти максимум всех dp[i]

for i in range(1, brain\_count + 1):

path\_length = max(path\_length, checked\_brain\_count[i])

return path\_length

n = 4

zbrain\_link\_list = [[] for i in range(n + 1)]

add\_brain\_link(zbrain\_link\_list, 3, 4, n)

add\_brain\_link(zbrain\_link\_list, 1, 2, n)

add\_brain\_link(zbrain\_link\_list, 1, 3, n)

add\_brain\_link(zbrain\_link\_list, 1, 4, n)

print(find\_longest\_path(zbrain\_link\_list, n))

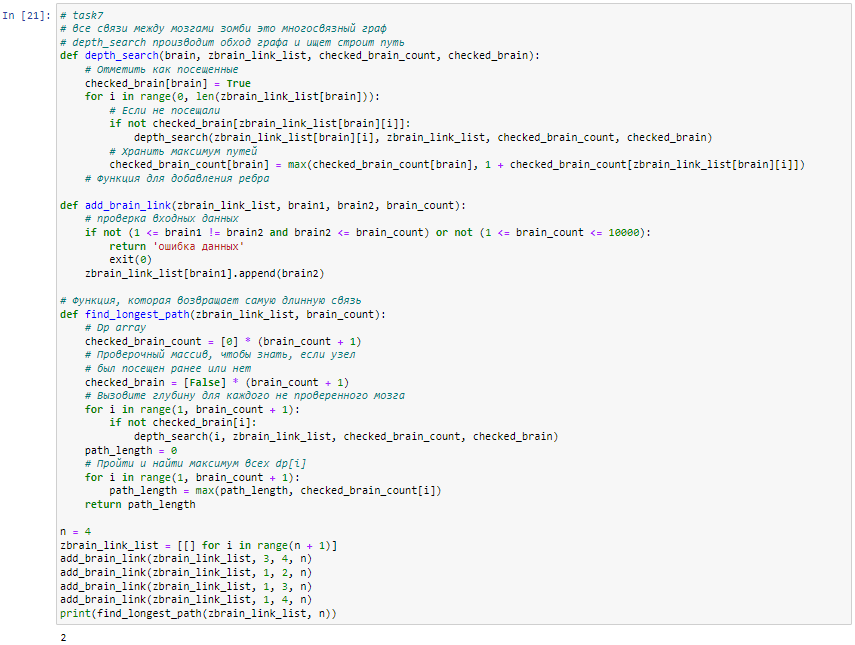


Рисунок 5 - результат работы программы

**Задача 8**

В перерывах между конкурсами топ-модель Изабелла старается не скучать и развиваться. Например, сейчас она учится собирать кубик Рубика 2x2x2.

Сразу научиться собирать кубик довольно непросто, поэтому она сначала хочет научиться понимать, можно ли из текущего положения собрать кубик поворотом ровно одной грани на 90 градусов в любую сторону.

Чтобы проверять правильность своих ответов, ей хочется иметь программу, которая будет по заданной раскраске граней кубика говорить, можно ли его собрать требуемым образом.

Кубик является собранным, если для каждой грани верно, что все квадратики этой грани имеют один цвет.

[https://ru.wikipedia.org/wiki/Кубик\_Рубика](https://ru.wikipedia.org/wiki/%D0%9A%D1%83%D0%B1%D0%B8%D0%BA_%D0%A0%D1%83%D0%B1%D0%B8%D0%BA%D0%B0)

Входные данные В первой строке дана последовательность из 24 целых чисел ai (1 ≤ ai ≤ 6), где ai — номер цвета i-го квадратика. Гарантируется, что каждый из цветов встречается ровно 4 раза.

Выходные данные Выведите «YES», если кубик можно собрать поворотом одной грани, и «NO» иначе.

Примеры  
входные данные  
2 5 4 6 1 3 6 2 5 5 1 2 3 5 3 1 1 2 4 6 6 4 3 4  
выходные данные  
NO  
входные данные  
5 3 5 3 2 5 2 5 6 2 6 2 4 4 4 4 1 1 1 1 6 3 6 3  
выходные данные  
YES

**Задача 9**

Задана последовательность a1, a2, ..., an одномерных отрезков, пронумерованных от 1 до n. Ваша задача — найти два различных индекса i и j таких, что отрезок ai лежит внутри отрезка aj.

Отрезок [l1, r1] лежит внутри отрезка [l2, r2] тогда и только тогда, когда l1 ≥ l2 и r1 ≤ r2.

Выведите позиции i и j. Если существует несколько возможных ответов, выведите любой из них. Если ответа не существует, то выведите -1 -1.

Входные данные В первой строке записано одно целое число n (1 ≤ n ≤ 3·105) — количество отрезков.

В каждой из следующих n строк записаны по два целых числа li и ri (1 ≤ li ≤ ri ≤ 109) — i-й отрезок.

Выходные данные Выведите две различные позиции i и j такие, что отрезок ai лежит внутри отрезка aj. Если существует несколько возможных ответов, выведите любой из них. Если ответа не существует, то выведите -1 -1.

Примеры  
входные данные  
5  
1 10  
2 9  
3 9  
2 3  
2 9  
выходные данные  
2 1

Решение задачи:

#task9  
s1 = [  
 "1 10",  
 "2 9",  
 "3 9",  
 "2 3",  
 "2 9"]  
  
def task9(n, s):  
 # перевод матрицы строк в матрицу чисел  
 if(n != len(s)):  
 return "Ошибка данных"  
 exit(0)  
  
 l = []  
 for i in range(0, n):  
 lt = s[i].split(" ")  
 row = []  
 for j in range(0, len(lt)): row.append(int(lt[j]))  
 l.append(row)  
  
 if not (1 <= n <= 3 \* 105):  
 return "Ошибка данных"  
 exit(0)  
  
 for i in range(0, n):  
 if not (1 <= l[i][0] <= l[i][1] <= 109):  
 return "Ошибка данных"  
 exit(0)  
 x = 0  
 if (n == len(l)):  
 for i in range(0, n):  
 for j in range(0, n):  
 if (l[i] > l[j] and l[i + 1] > l[j + 1]):  
 x = 1  
 return i+1, j+1  
 elif (l[i] > l[j] and l[i + 1] > l[j + 1]):  
 x = 1  
 return j+1, i+1  
 else:  
 return "-1 -1"  
  
print(task9(5, s1))



Рисунок 6 - результат работы программы

**Задача 10**

Автобус курсирует по координатной прямой Ox от точки x = 0 до точки x = a. Выехав из точки x = 0, он доезжает до точки x = a, сразу разворачивается и едет обратно в точку x = 0. Вернувшись в точку x = 0, он сразу снова едет в точку x = a и так далее. Таким образом, автобус курсирует от x = 0 до x = a и обратно. Переезд из точки x = 0 в x = a или из точки x = a в x = 0 называется рейсом автобуса. Всего автобус должен совершить k рейсов.

Бензобак автобуса вмещает b литров бензина. На преодоление единицы расстояния автобус тратит ровно один литр бензина. Свой путь автобус начинает с полным бензобаком.

Между x = 0 и x = a в точке x = f находится заправка. Других заправок на пути автобуса нет. Проезжая мимо заправки в любом из двух направлений, автобус может остановиться и пополнить запас бензина в бензобаке. Таким образом, после остановки на заправку в бензобаке автобуса будет b литров бензина.

Какое минимальное количество раз автобус должен дозаправиться в точке x = f, чтобы начав путь в точке x = 0 совершить k рейсов?

Входные данные В первой строке записаны четыре целых числа a, b, f, k (0 < f < a ≤ 106, 1 ≤ b ≤ 109, 1 ≤ k ≤ 104) — точка окончания первого автобусного рейса, вместимость бензобака автобуса, точка, в которой расположена заправка, а также требуемое количество рейсов.

Выходные данные Выведите минимальное количество дозаправок автобуса, которые необходимы, чтобы совершить k рейсов. Если автобус не сможет совершить k рейсов, выведите -1.

Примеры  
входные данные  
6 9 2 4  
выходные данные  
4

Решение задачи:

def task10(a, b, f, k):  
 v = b # бензина в баке  
 x = 0 # счётчик заправок  
  
 if b<2\*f or b < 2\*(a - f):  
 return -1  
 for i in range(0, k):  
 if i % 2: # не четный рейс  
 v = v - a + f # доехали до заправки в обратном направлении  
 if v < f + f: # проверяем, что бензина не хватит доехать до конечной и вернуться  
 v = b # заправились  
 x = x + 1  
 v = v - f # доехали до конечной  
 else: # четный рейс  
 v = v - f # доехали до заправки в прямом направлении  
 if v < (a - f) \* 2: # проверяем надо ли заправится  
 v = b # заправились  
 x = x + 1  
 v = v - a + f # доехали до конечной  
  
 return x  
  
  
a = 6 # длинна рейса  
b = 9 # емкость бака  
f = 2 # растояние до заправки  
k = 4 # число рейсов  
  
print(task10(a, b, f, k))



Рисунок 7 – результат работы программы

**Заключение**

В ходе выполнения курсовой работы было решён ряд задач в соответствии с вариантом, поэтому курсовая работа является выполненной.

**Приложение**

# task1

# s = input("Введите c и d через пробел:")

s1 = '1 10'

# s = input("Введите n и m через пробел:")

s2 = '7 2'

# s = input("Введите k:")

s3 = '1 3'

def task1 (s1, s2, s3):

s1 = s1.split(" ")

if(len(s1) > 2):

return 'ошибка данных'

exit(0)

c = int(s1.pop(0))

d = int(s1.pop(0))

if not (1 <= c <= 100) or not (1 <= d <= 100):

return 'ошибка данных'

exit(0)

# s = input("Введите n и m через пробел:")

s2 = s2.split(" ")

if (len(s2) > 2):

return 'ошибка данных'

exit(0)

n = int(s2.pop(0))

m = int(s2.pop(0))

if not (1 <= n <= 100) or not (1 <= m <= 100):

return 'ошибка данных'

exit(0)

# s = input("Введите k:")

s3 = s3.split(" ")

if (len(s3) > 1):

return 'ошибка данных'

exit(0)

k = int(s3.pop(0))

if not (1 <= k <= 100):

return 'ошибка данных'

exit(0)

total\_winners = n \* m # всего финалистов

# требуемое количество победтелей

need\_winners = total\_winners - k

# если за один,основной, раунд соревнований нельзя закрыть все требуемые позиции - 1(победитель дополнительного раунда), то необходимо провести еще один основной раунд

# если проводится дополнительный раунд соревнований, то приоритет у победителей основного раунда

main\_round\_count = 0 # количество основных раундов

while int((need\_winners) - (main\_round\_count \* n)) >= main\_round\_count:

main\_round\_count = main\_round\_count + 1

task\_count = 0

if need\_winners - main\_round\_count \* n <= main\_round\_count:

task\_count = c \* main\_round\_count

return task\_count

else:

task\_count = c \* main\_round\_count + d \* (need\_winners - main\_round\_count \* n)

return task\_count

print(task1(s1, s2, s3))

#task2

n = 3

s1 = '110000'

s2 = '000111'

# проверить слово на валидность

def check\_word(s):

for i in range(0, len(s)):

if s[i] != '1' and s[i] != '0': return 0

return 1

# конвертировать строку в список

def convert\_string\_to\_list(str):

list = []

for i in range(0, len(str)):

list.append(str[i])

return list

# определить победителя

def calculate\_result(list1, list2):

if list1.count('1') > list2.count('1'):

return "First"

elif list1.count('1') < list2.count('1'):

return "Second"

else:

return "Draw"

if not (1 <= n <= 106) or int(n) \* 2 != len(s1) or int(n) \* 2 != len(s2) or not check\_word(s1) or not check\_word(s2):

return "Ошибка данных"

exit(0)

s1 = convert\_string\_to\_list(s1)

s2 = convert\_string\_to\_list(s2)

l1 = [] # ходы Ярослава

l2 = [] # ходы Андрея

# чтобы победить каждый из игроков должен набоать максимально возможное количество "1"

for i in range(0, n):

# ход Ярослава

if s1.count('1') > 0:

l1.append(s1.pop(s1.index('1')))

else:

l1.append(s1.pop())

# ход Андрея

if s2.count('1') > 0:

l2.append(s2.pop(s2.index('1')))

else:

l2.append(s2.pop())

print(calculate\_result(l1, l2))

n = 5

s1 = "1 2 3 4 5"

s2 = "1 5 2 3 4"

# конвертировать строку в список

def convert\_string\_to\_list(str):

list = []

str = str.split(" ")

for i in range(0, len(str)):

list.append(int(str[i]))

return list

# проверить исходные данные

def validate\_data(list1, list2, n):

if not (1 <= n <= 2 \* 105) or len(list1) != len(list2) or len(list1) != n:

return False

l1 = list1.copy()

l2 = list2.copy()

l1.sort()

l2.sort()

# отсортированные списки должны быть идентиичны

if l1 != l2:

return False

# списки должны быть от 1 до n

for i in range(1, n):

if l1[i] != l1[i - 1] + 1:

return False

return True

# функция ищет кратчайщее пребразование из l1 в l2

# иначе начинаем с начала списка

# ищем 2 идущих по порядку числа и вставляем последнее число списка между ними

# далее рекурсивно вызываем функцию

# так проделываем для всех пар чисел списка на первом уровне

def find\_answer(l1, l2, layer):

# делаем искуственное ограничение на глубину рекурсии равную 2 \* n

# то есть, если решение не найдено за 2n шагов, то дальще искать не имеет смысла

if layer > len(l1) \* 2:

return len(l1) \* 2

# если l1 и l2 равны возвращаем layer - уровень рекурсии(колличество сделанных операций)

# if compare\_list(l1, l2):

if l1 == l2:

return layer

# вставляем последнее число перед первым

ll = l1.copy()

ll.insert(0, ll.pop(len(ll)-1))

# и передаем в рекурсивно в саму себя

min\_layer = find\_answer(ll, l2, layer + 1)

# последовательно вставляем последний элемент списка

# между всеми парами идущими по порядку

for i in range(1, len(l2)):

ll = l1.copy()

ll.insert(i, ll.pop(len(ll)-1))

min\_layer = min(min\_layer, find\_answer(ll, l2, layer + 1))

return min\_layer

# конвертируем исходные данные в список

start\_list = convert\_string\_to\_list(s1)

destination\_list = convert\_string\_to\_list(s2)

# проверяем ИД на соответствие условиям задачи

if not validate\_data(start\_list, destination\_list, n):

print("Исходные данные не отвечают условиям задачи")

exit(0)

print(find\_answer(start\_list, destination\_list, 0))

in\_data = [

"ahmed",

3,

"ahmed posted on fatma's wall",

"fatma commented on ahmed's post",

"mona likes ahmed's post"]

# функция сортировки списка

# пузырьком

def sort\_list(l):

for i in range(1, len(l)):

for j in range(i, len(l)):

if l[j][1] > l[j - 1][1] or (l[j][1] == l[j - 1][1] and l[j-1][0] > l[j][0]):

buf = l[j]

l[j] = l[j - 1]

l[j - 1] = buf

return l

# ищем в списке имя и увеличиваем вес или создаем запись

def append\_to\_list(name, weight, list):

fnd = 0

for i in range(0, len(list)): # ищем в списке результатов имя друга для увеличения его коэффициента

if list[i][0] == name:

list[i][1] = list[i][1] + weight

fnd = 1

break # нашли продолжать цикл нет необходимости

if not fnd: # если не найден, то добавляем

list.append([name, weight])

return list

# функция проверяет входные данные

# и формирует список коэффициентов дружбы по каждой уникальной персоне

# входным параметром являются все исходные данные

def make\_data(list):

score\_count = list[1] # количество оценок в переданном списке

if score\_count != len(list) - 2:

return False

my\_name = list[0] # мое имя

friend\_list = [] # список коэффициентов дружбы по всем оценкам

for i in range(2, len(list)):

s = list[i].split(" ") # разбираем строку с оценкой на составляющие

temp\_list = []

if s[1] == "posted" and s[2] == "on" and s[4] == "wall" and len(s) == 5:

name1 = s[0]

name2 = s[3].split("'")[0]

if name1 == my\_name or name2 == my\_name:

weight = 15

else:

weight = 0

elif s[1] == "commented" and s[2] == "on" and s[4] == "post" and len(s) == 5:

name1 = s[0]

name2 = s[3].split("'")[0]

if name1 == my\_name or name2 == my\_name:

weight = 10

else:

weight = 0

elif s[1] == "likes" and s[3] == "post" and len(s) == 4:

name1 = s[0]

name2 = s[2].split("'")[0]

if name1 == my\_name or name2 == my\_name:

weight = 5

else:

weight = 0

else:

return False

friend\_list.append([name1, name2, weight]) # сохраняем оценку

result\_list = [] # список коэффициентов дружбы

for i in range(0, len(friend\_list)):

if friend\_list[i][0] == my\_name:

result\_list = append\_to\_list(friend\_list[i][1], friend\_list[i][2], result\_list)

elif friend\_list[i][1] == my\_name:

result\_list = append\_to\_list(friend\_list[i][0], friend\_list[i][2], result\_list)

elif friend\_list[i][0] != my\_name and friend\_list[i][1] != my\_name and friend\_list[i][0] != friend\_list[i][1]:

result\_list = append\_to\_list(friend\_list[i][0], friend\_list[i][2], result\_list)

result\_list = append\_to\_list(friend\_list[i][1], friend\_list[i][2], result\_list)

else:

return False

result\_list = sort\_list(result\_list)

for\_return = []

# готовим к возврату только имена без весов отсортированные в порядке уменьшения коэффициента дружбы и алфавита

for i in range(0, len(result\_list)):

for\_return.append(result\_list.pop(0)[0])

return for\_return

result = make\_data(in\_data)

if not result:

return "Ошибка данных"

print(result)

# task7

# все связи между мозгами зомби это многосвязный граф

# depth\_search производит обход графа и ищет строит путь

def depth\_search(brain, zbrain\_link\_list, checked\_brain\_count, checked\_brain):

# Отметить как посещенные

checked\_brain[brain] = True

for i in range(0, len(zbrain\_link\_list[brain])):

# Если не посещали

if not checked\_brain[zbrain\_link\_list[brain][i]]:

depth\_search(zbrain\_link\_list[brain][i], zbrain\_link\_list, checked\_brain\_count, checked\_brain)

# Хранить максимум путей

checked\_brain\_count[brain] = max(checked\_brain\_count[brain], 1 + checked\_brain\_count[zbrain\_link\_list[brain][i]])

# Функция для добавления ребра

def add\_brain\_link(zbrain\_link\_list, brain1, brain2, brain\_count):

# проверка входных данных

if not (1 <= brain1 != brain2 and brain2 <= brain\_count) or not (1 <= brain\_count <= 10000):

return 'ошибка данных'

exit(0)

zbrain\_link\_list[brain1].append(brain2)

# Функция, которая возвращает самую длинную связь

def find\_longest\_path(zbrain\_link\_list, brain\_count):

# Dp array

checked\_brain\_count = [0] \* (brain\_count + 1)

# Проверочный массив, чтобы знать, если узел

# был посещен ранее или нет

checked\_brain = [False] \* (brain\_count + 1)

# Вызовите глубину для каждого не проверенного мозга

for i in range(1, brain\_count + 1):

if not checked\_brain[i]:

depth\_search(i, zbrain\_link\_list, checked\_brain\_count, checked\_brain)

path\_length = 0

# Пройти и найти максимум всех dp[i]

for i in range(1, brain\_count + 1):

path\_length = max(path\_length, checked\_brain\_count[i])

return path\_length

n = 4

zbrain\_link\_list = [[] for i in range(n + 1)]

add\_brain\_link(zbrain\_link\_list, 3, 4, n)

add\_brain\_link(zbrain\_link\_list, 1, 2, n)

add\_brain\_link(zbrain\_link\_list, 1, 3, n)

add\_brain\_link(zbrain\_link\_list, 1, 4, n)

print(find\_longest\_path(zbrain\_link\_list, n))

#task9

s1 = [

"1 10",

"2 9",

"3 9",

"2 3",

"2 9"]

def task9(n, s):

if(n != len(s)):

return "Ошибка данных"

exit(0)

# перевод матрицы строк в матрицу чисел

l = []

for i in range(0, n):

lt = s[i].split(" ")

row = []

for j in range(0, len(lt)): row.append(int(lt[j]))

l.append(row)

if not (1 <= n <= 3 \* 105):

return "Ошибка данных"

exit(0)

for i in range(0, n):

if not (1 <= l[i][0] <= l[i][1] <= 109):

return "Ошибка данных"

exit(0)

x = 0

if (n == len(l)):

for i in range(0, n):

for j in range(0, n):

if (l[i] > l[j] and l[i + 1] > l[j + 1]):

x = 1

return i+1, j+1

elif (l[i] > l[j] and l[i + 1] > l[j + 1]):

x = 1

return j+1, i+1

else:

return "-1 -1"

print(task9(5, s1))

#task10

def task10(a, b, f, k):

v = b # бензина в баке

x = 0 # счётчик заправок

#проверка входных данных

if not (0 < f < a <= 106 and 1 <= b <= 109 and 1 <= k <= 104):

return "Ошибка данных"

exit(0)

if b<2\*f or b < 2\*(a - f):

return -1

for i in range(0, k):

if i % 2: # не четный рейс

v = v - a + f # доехали до заправки в обратном направлении

if v < f + f: # проверяем, что бензина не хватит доехать до конечной и вернуться

v = b # заправились

x = x + 1

v = v - f # доехали до конечной

else: # четный рейс

v = v - f # доехали до заправки в прямом направлении

if v < (a - f) \* 2: # проверяем надо ли заправится

v = b # заправились

x = x + 1

v = v - a + f # доехали до конечной

return x

a = 6 # длинна рейса

b = 9 # емкость бака

f = 2 # растояние до заправки

k = 4 # число рейсов

print(task10(a, b, f, k))