

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет: Информатика и вычислительная техника

Кафедра: КБИС

**Отчёт по лабораторным работам**

**по дисциплине “Методы программирования”**

Выполнил – ст. гр. ВКБ31 Котелевец К.А.

Ростов-на-Дону

2025

**Лабораторная работа 1**

**Задача 1: Гоблины и шаманы**

Гоблины Мглистых гор очень любях ходить к своим шаманам. Так как гоблинов много, к шаманам часто образуются очень длинные очереди. А поскольку много гоблинов в одном месте быстро образуют шумную толку, которая мешает шаманам проводить сложные медицинские манипуляции, последние решили установить некоторые правила касательно порядка в очереди.

Обычные гоблины при посещении шаманов должны вставать в конец очереди. Привилегированные же гоблины, знающие особый пароль, встают ровно в ее середину, причем при нечетной длине очереди они встают сразу за центром.

Так как гоблины также широко известны своим непочтительным отношением ко всяческим правилам и законам, шаманы попросили вас написать программу, которая бы отслеживала порядок гоблинов в очереди.

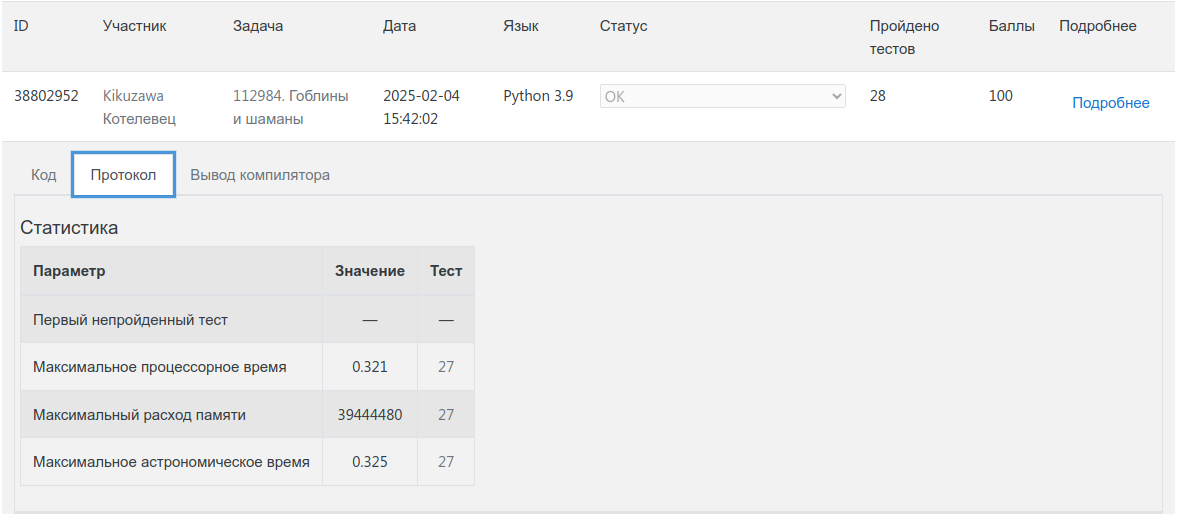
**Входные данные**

В первой строке входных данный записано число N (1≤N≤105) - количество запросов к программе. Следующие N строк содержат описание запросов в формате:

* "+ i" - гоблин с номером i (1≤i≤N) встает в конец очереди.
* "\* i" - привилегированный гоблин с номером i встает в середину очереди.
* "-" - первый гоблин из очереди уходит к шаманам. Гарантируется, что на момент такого запроса очередь не пуста.

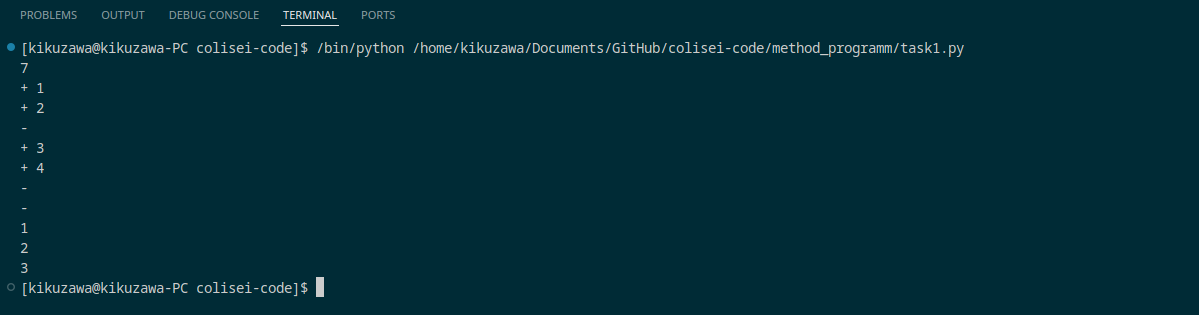
#### Выходные данные

Для каждого запроса типа "-" программа должна вывести номер гоблина, который должен зайти к шаманам.



Листинг программы:

|  |
| --- |
| from collections import deque  from typing import Sequence, List, Iterable  **def** process\_queue(actions: Sequence[Sequence[str]]) -> Iterable[str]:  q1 = deque()  q2 = deque()  results: List[str] = []  for action in actions:  if action[0] == '+':  q2.append(action[1])  elif action[0] == '\*':  q2.appendleft(action[1])  else:  results.append(q1.popleft())  if len(q1) < len(q2):  q1.append(q2.popleft())  return results  **def** main() -> None:  n: int = int(input())  actions: List[List[str]] = []  for \_ in range(n):  action: List[str] = input().split()  actions.append(action)  results: Iterable[str] = process\_queue(actions)  for result in results:  print(result)  if \_\_name\_\_ == "\_\_main\_\_":  main() |



**Задача 2: Баржа**

На барже располагается K грузовых отсеков. В каждый отсек можно поместить некоторое количество бочек с одним из 10 000 видов топлива. Причём извлечь бочку из отсека можно лишь в случае, если все бочки, помещённые в этот отсек после неё, уже были извлечены. Таким образом в каждый момент времени в каждом непустом отсеке имеется ровно одна бочка, которую можно извлечь не трогая остальных. Будем называть такие бочки крайними.

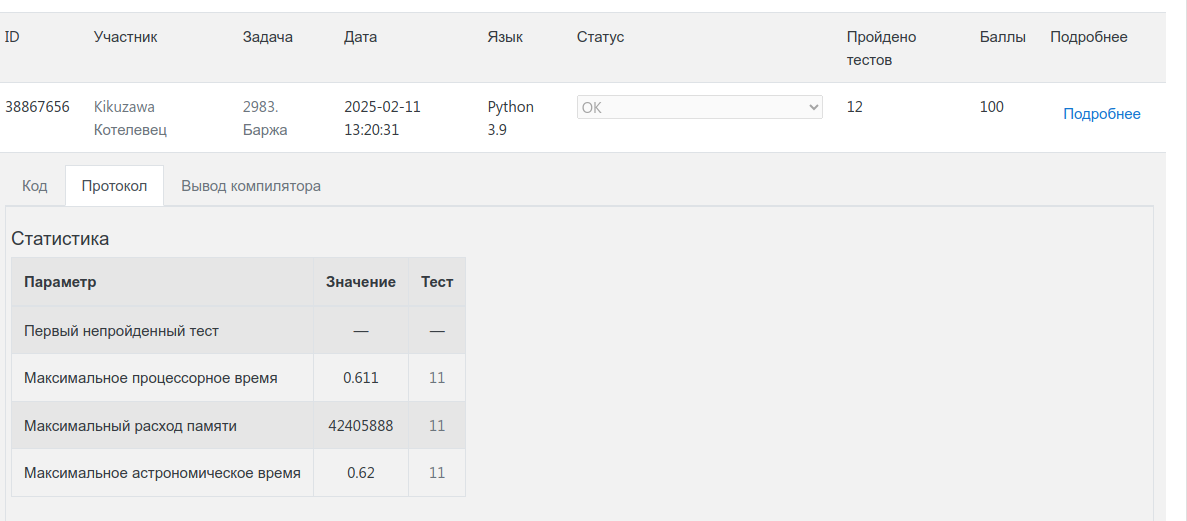
Изначально баржа пуста. Затем она последовательно проплывает через N доков, причём в каждом доке на баржу либо погружается бочка с некоторым видом топлива в некоторый отсек, либо выгружается крайняя бочка из некоторого отсека. Однако, если указанный отсек пуст, либо если выгруженная бочка содержит не тот вид топлива, который ожидалось, следует зафиксировать ошибку. Если на баржу оказывается погружено более P бочек или если после прохождения всех доков она не стала пуста, следует также зафиксировать ошибку. От вас требуется либо указать максимальное количество бочек, которые одновременно пребывали на барже либо зафиксировать ошибку.

**Входные данные**

В первой строке три целых числа N, K и P (1 ≤ N, K, P ≤ 100 000). Далее следует N строк с описанием действия, выполняемого в очередном доке. Если в нём происходит погрузка, то строка имеет вид «+ A B», где A — номер отсека, в который помещается бочка, а B — номер вида топлива в ней. Если же док занимается разгрузкой, то строка имеет вид «- A B», где A — номер отсека, из которого извлекается бочка, а B — номер ожидаемого вида топлива.

**Выходные данные**

Вывести либо одно число, равное искомому максимуму в случае безошибочного прохождения баржой маршрута, либо вывести слово «Error» в противном случае.



Листинг программы:

|  |
| --- |
| from collections import deque  from typing import List, Deque, Iterable  from dataclasses import dataclass  @dataclass(frozen=True)  class Action:  operation: str # Операция: "+" для погрузки, "-" для разгрузки  cell\_index: int # Номер отсека (1-based индекс)  fuel\_type: str # Тип топлива в бочке  def process\_docks(cells: int, max\_len: int, actions: Iterable[Action]):  max\_tanks: int = 0 # Максимальное количество бочек на барже  current\_tanks: int = 0 # Текущее количество бочек на барже  error: bool = False # Флаг ошибки  # Инициализация отсеков: каждый отсек представлен деком (очередью)  cell: List[Deque[str]] = [deque() for \_ in range(cells)]  # Обработка каждого действия  for action in actions:  if action.operation == "+": # Погрузка бочки  cell\_index: int = action.cell\_index - 1 # Преобразуем в 0-based индекс  cell[cell\_index].append(action.fuel\_type) # Добавляем бочку в отсек  current\_tanks += 1 # Увеличиваем счетчик бочек  # Проверка на превышение максимального количества бочек  if current\_tanks > max\_len:  error = True  break  # Обновляем максимальное количество бочек  max\_tanks = max(max\_tanks, current\_tanks)  else: # Разгрузка бочки  cell\_index: int = action.cell\_index - 1 # Преобразуем в 0-based индекс  # Проверка на пустой отсек или несоответствие типа топлива  if not cell[cell\_index] or cell[cell\_index].pop() != action.fuel\_type:  error = True  break  current\_tanks -= 1 # Уменьшаем счетчик бочек  # Возвращаем -1, если баржа не пуста или произошла ошибка, иначе возвращаем максимальное количество бочек  return -1 if current\_tanks > 0 or error else max\_tanks  def main():  # Чтение входных данных  request: List[str] = input().split()  docks: int = int(request[0]) # Количество доков  cells: int = int(request[1]) # Количество отсеков  max\_len: int = int(request[2]) # Максимальное количество бочек на барже  # Создание списка действий  actions: List[Action] = []  for \_ in range(docks):  action\_data: List[str] = input().split()  action = Action(operation=action\_data[0], cell\_index=int(action\_data[1]), fuel\_type=action\_data[2])  actions.append(action)  # Обработка действий и получение результата  result: int = process\_docks(cells, max\_len, actions)  # Вывод результата  print("Error" if result == -1 else result)  main() |

**Задание 3. Реклама**

Фирма NNN решила транслировать свой рекламный ролик в супермаркете XXX. Однако денег, запланированных на рекламную кампанию, хватило лишь на две трансляции ролика в течение одного рабочего дня. И при этом обязательно транслировать ровно два рекламных ролика в день.

Фирма собрала информацию о количестве покупателей в каждый момент некоторого дня. Менеджер по рекламе предположил, что и на следующий день покупатели будут приходить и уходить ровно в те же моменты времени.

Помогите ему определить моменты времени, когда нужно включить трансляцию рекламных роликов, чтобы как можно большее количество покупателей прослушало ролик.

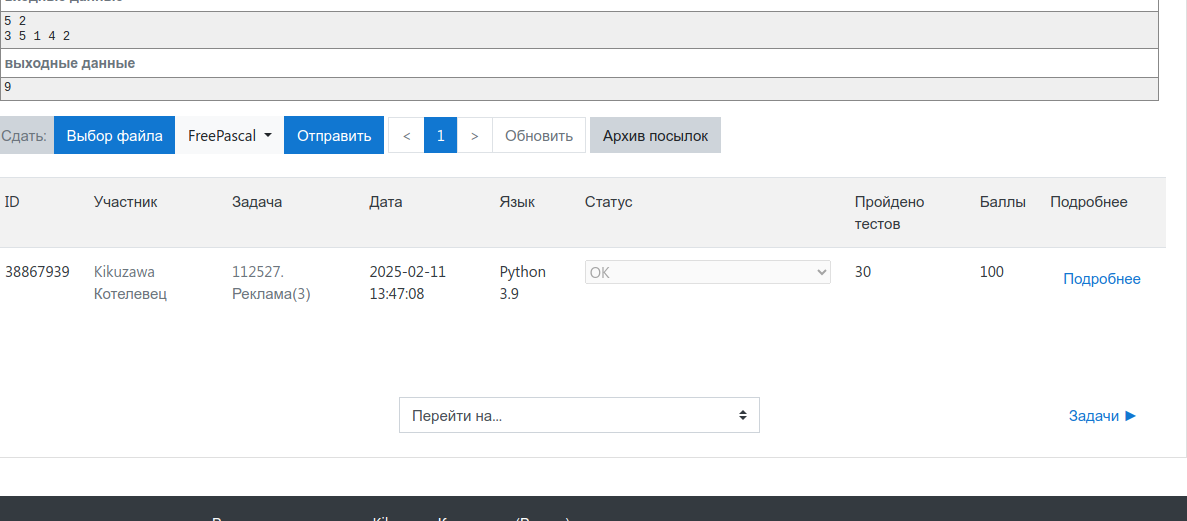
Ролик длится ровно одну единицу времени. Для каждого момента времени известно количество покупателей, находящихся в магазине в этот момент. Между концом первой рекламы и началом следующей должна пройти как минимум *К*- 1 единица времени.

**Входные данные**

Первая строка содержит два числа *N*(моментов времени) и *K*(время совершения покупки одним покупателем). Гарантируется, что *N*> *K.*( *N, K*≤ 200 000 ). Во второй строке содержится *N*чисел *a i*— количество покупателей в момент времени *i*. ( 0 ≤ *a i*≤ 2 000 000 000 ).

**Выходные данные**

Выведите единственное число — количество просмотревших рекламу покупателей.



Листинг программы

|  |
| --- |
| from itertools import combinations, product  from typing import List, Set, Iterable, Tuple  def find\_best\_broadcast(n: int, k: int, customer\_counts: List[int]) -> int:  """  Находит максимальное количество покупателей, которые увидят два рекламных ролика,  транслируемых с учетом ограничений на время между показами.  :param n: Количество моментов времени.  :param k: Минимальное время между окончанием первого ролика и началом второго.  :param customer\_counts: Список количества покупателей в каждый момент времени.  :return: Максимальное количество покупателей, которые увидят оба ролика.  """  # Находим максимальное количество покупателей в любой момент времени  max\_customers: int = max(customer\_counts)  # Находим все моменты времени, когда количество покупателей максимально  max\_customers\_times: Set[int] = {index for index, value in enumerate(customer\_counts)  if value == max\_customers}  # Проверяем, есть ли два момента с максимальным количеством покупателей,  # между которыми прошло достаточно времени  for time1, time2 in combinations(max\_customers\_times, 2):  if abs(time1 - time2) > k - 1:  return max\_customers \* 2 # Если да, возвращаем удвоенное максимальное значение  # Если таких моментов нет, ищем пару (максимальный момент + другой момент),  # где между ними прошло достаточно времени  all\_possible\_pairs: Iterable[Tuple[int, int]] = product(  max\_customers\_times,  range(len(customer\_counts))  )  # Находим второе по величине количество покупателей, которое можно использовать  # в паре с максимальным, соблюдая ограничение по времени  second\_max\_customers: int = max(  map(  lambda pair: customer\_counts[pair[1]],  filter(  lambda pair: pair[0] != pair[1] and abs(pair[0] - pair[1]) > k - 1,  all\_possible\_pairs  )  ),  default=0  )  # Если нашли подходящую пару, возвращаем сумму максимального и второго по величине  if second\_max\_customers > 0:  return max\_customers + second\_max\_customers  # Если не нашли подходящих пар, перебираем все возможные пары моментов времени  # с учетом ограничения на минимальное время между показами  max\_viewers\_sum: int = 0  for first\_ad\_time in range(n - k):  second\_ad\_time: int = first\_ad\_time + k  if second\_ad\_time < n:  viewers\_sum: int = customer\_counts[first\_ad\_time] + max(customer\_counts[second\_ad\_time:n])  max\_viewers\_sum = max(max\_viewers\_sum, viewers\_sum)  return max\_viewers\_sum  def main():  """  Основная функция для ввода данных и вывода результата.  """  # Ввод данных  n, k = map(int, input().split())  customer\_counts: List[int] = list(map(int, input().split()))    # Вычисление результата  result: int = find\_best\_broadcast(n, k, customer\_counts)    # Вывод результата  print(result)  if \_\_name\_\_ == "\_\_main\_\_":  main() |

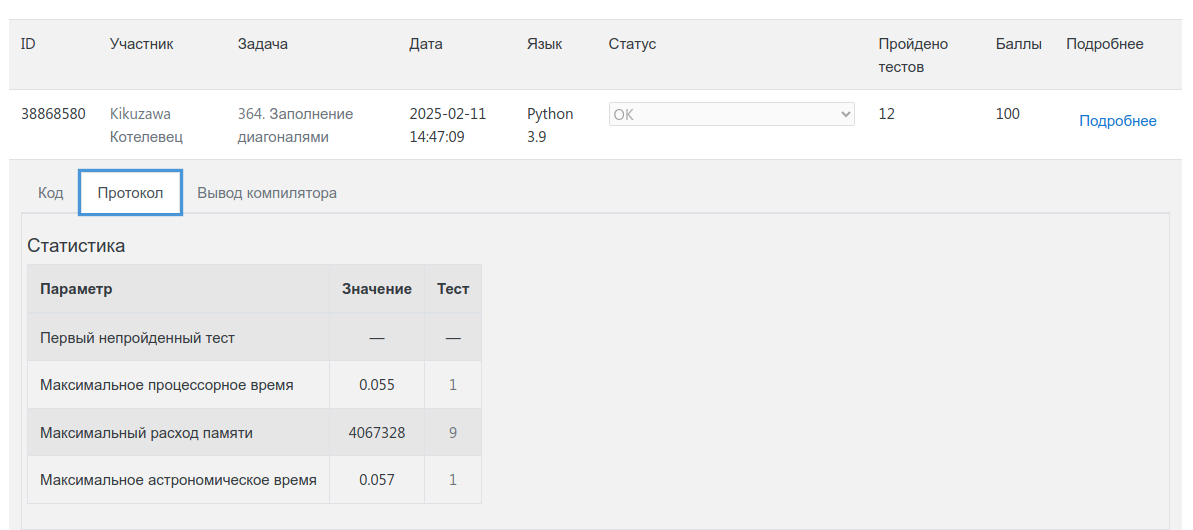
**Задание 4.**  **Заполнение диагоналями**  
 Даны числа n и m. Создайте массив A[n][m] и заполните его, как показано на примере.

**Входные данные**

Программа получает на вход два числа n и m.

**Выходные данные**

Программа должна вывести  полученный массив.



Листинг программы

|  |
| --- |
| from typing import List  def fill\_matrix(n: int, m: int) -> List[List[int]]:  """  Создает и заполняет двумерный массив (матрицу) размером n x m по диагоналям.  :param n: Количество строк в матрице.  :param m: Количество столбцов в матрице.  :return: Заполненная матрица.  """  # Создаем пустую матрицу размером n x m, заполненную нулями  matrix: List[List[int]] = [[0 for \_ in range(m)] for \_ in range(n)]  # Счетчик для заполнения матрицы значениями  count: int = 0  # Перебираем все диагонали матрицы  for index\_of\_diagonal in range(n + m - 1):  # Определяем начальную точку для заполнения текущей диагонали  start\_col: int  start\_row: int  if index\_of\_diagonal < m:  # Если диагональ начинается в верхней строке  start\_col = index\_of\_diagonal  start\_row = 0  else:  # Если диагональ начинается в последнем столбце  start\_col = m - 1  start\_row = index\_of\_diagonal - m + 1  # Заполняем текущую диагональ  while start\_col >= 0 and start\_row < n:  matrix[start\_row][start\_col] = count  count += 1  start\_col -= 1 # Двигаемся влево по столбцам  start\_row += 1 # Двигаемся вниз по строкам  return matrix  def main():  """  Основная функция для ввода данных и вывода результата.  """  # Ввод данных: количество строк (n) и столбцов (m)  n, m = map(int, input().strip().split())    # Заполнение матрицы  matrix = fill\_matrix(n, m)    # Вывод матрицы  for row in matrix:  # Форматированный вывод каждой строки матрицы  print(' '.join(f'{num:3d}' for num in row))  if \_\_name\_\_ == "\_\_main\_\_":  main() |

**Задание 5. Правильная скобочная последовательность**

Рассмотрим последовательность, состоящую из круглых, квадратных и фигурных скобок. Программа дожна определить, является ли данная скобочная последовательность правильной.

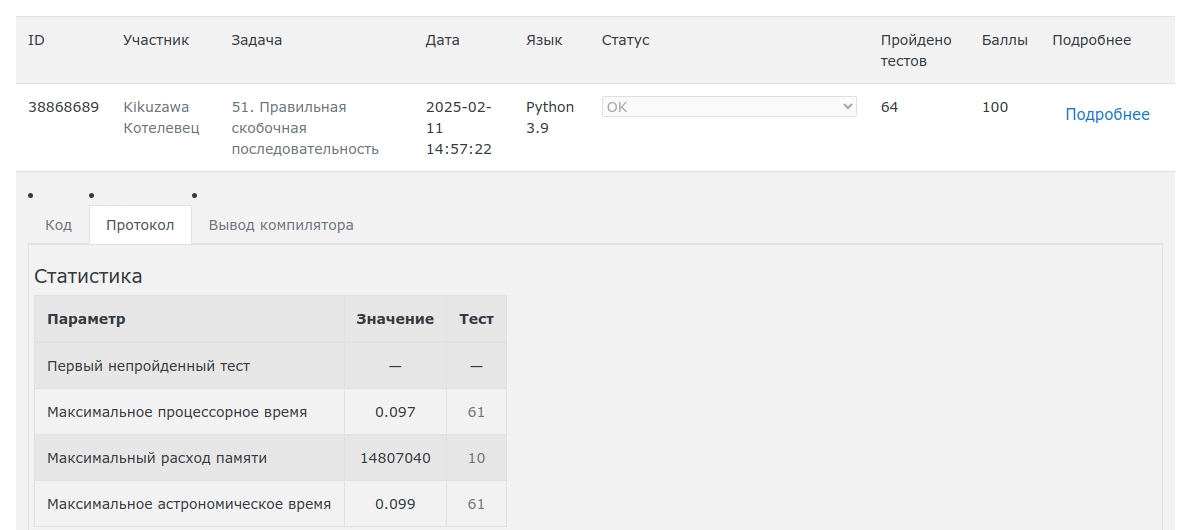
Пустая последовательность явлется правильной. Если A – правильная, то последовательности (A), [A], {A} – правильные. Если A и B – правильные последовательности, то последовательность AB – правильная.

**Входные данные**

В единственной строке записана скобочная последовательность, содержащая не более 100000 скобок.

**Выходные данные**

Если данная последовательность правильная, то программа должна вывести строку yes, иначе строку no.



Листинг программы:

|  |
| --- |
| # Импортируем необходимые модули для работы со стеком и типами данных  from collections import deque  from typing import Dict  def is\_balanced(sequence):  """  Проверяет, является ли скобочная последовательность правильной согласно условиям задачи:  - пустая последовательность считается правильной  - если A правильная, то (A), [A], {A} тоже правильные  - если A и B правильные, то AB тоже правильное    Args:  sequence (str): строка из скобок длиной не более 100000 символов    Returns:  bool: True если последовательность правильная, False иначе  """  # Создаем стек для хранения открывающих скобок  stack: deque[str] = deque()    # Словарь соответствия закрывающих и открывающих скобок  matching\_brackets: Dict[str, str] = {')': '(', '}': '{', ']': '['}  # Проходим по каждому символу в последовательности  for char in sequence:  # Если встретили открывающую скобку - добавляем её в стек  if char in matching\_brackets.values():  stack.append(char)  # Если встретили закрывающую скобку  elif char in matching\_brackets:  # Проверяем, что стек не пустой и верхняя скобка совпадает с текущей  if not (stack and stack[-1] == matching\_brackets[char]):  return False # Последовательность неправильная  stack.pop() # Убираем соответствующую открывающую скобку из стека  # Последовательность правильная, если стек пуст (все скобки закрыты)  return not bool(stack)  def main():  """  Основная функция программы для обработки входных данных.  Читает строку из скобок и выводит результат проверки.  """  # Читаем последовательность скобок из входного потока  sequence = input()    # Выводим результат в требуемом формате  print("yes" if is\_balanced(sequence) else "no")  # Запускаем программу  main() |

**Задание 6. Сортировка вагонов**

К тупику со стороны пути 1 (см. рисунок) подъехал поезд. Разрешается отцепить от поезда один или сразу несколько первых вагонов и завезти их в тупик (при желании, можно даже завезти в тупик сразу весь поезд). После этого часть из этих вагонов вывезти в сторону пути 2. После этого можно завезти в тупик еще несколько вагонов и снова часть оказавшихся вагонов вывезти в сторону пути 2. И так далее (так, что каждый вагон может лишь один раз заехать с пути 1 в тупик, а затем один раз выехать из тупика на путь 2). Заезжать в тупик с пути 2 или выезжать из тупика на путь 1 запрещается. Нельзя с пути 1 попасть на путь 2, не заезжая в тупик.

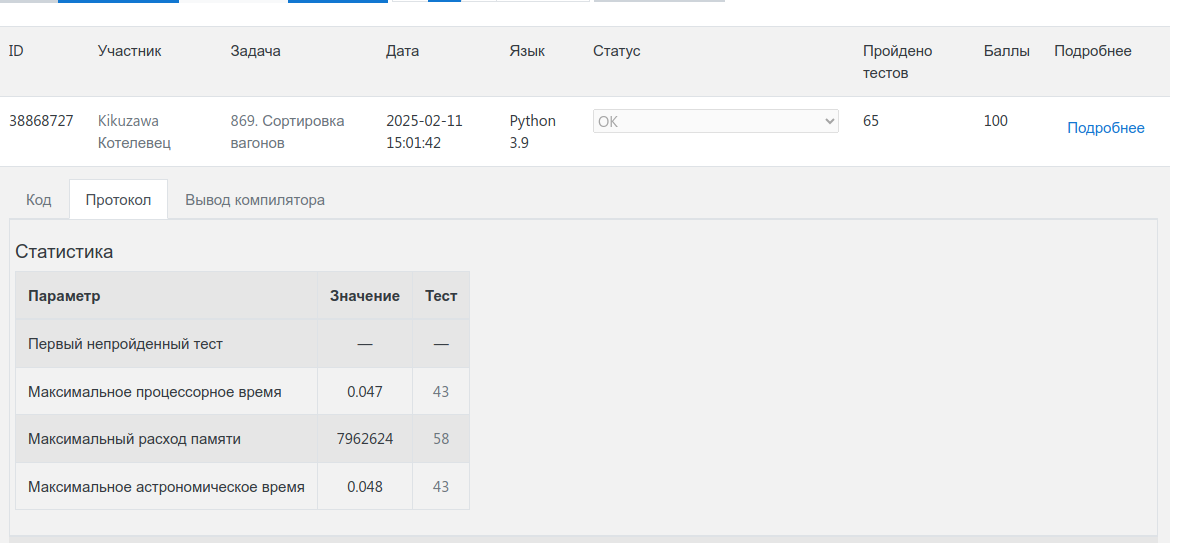
Известно, в каком порядке изначально идут вагоны поезда. Требуется с помощью указанных операций сделать так, чтобы вагоны поезда шли по порядку (сначала первый, потом второй и т.д., считая от головы поезда, едущего по пути 2 в сторону от тупика). Напишите программу, определяющую, можно ли это сделать.

**Входные данные**

Вводится число *N* — количество вагонов в поезде (1≤*N*≤100). Дальше идут номера вагонов в порядке от головы поезда, едущего по пути 1 в сторону тупика. Вагоны пронумерованы натуральными числами от 1 до *N*, каждое из которых встречается ровно один раз.

**Выходные данные**

Если сделать так, чтобы вагоны шли в порядке от 1 до *N*, считая от головы поезда, когда поезд поедет по пути 2 из тупика, можно, выведите сообщение YES, если это сделать нельзя, выведите NO.



Листинг программы:

|  |
| --- |
| from collections import deque  from typing import List, Iterable  def can\_sort\_train(n, wagons):  # Используем deque для стека тупика и стека пути 2  dead\_end\_stack: deque[int] = deque() # Стек для тупика  output\_track\_stack: deque[int] = deque() # Стек для пути 2  for wagon in wagons:  # Перемещаем вагоны из тупика на путь 2, если следующий вагон в порядке  while dead\_end\_stack and dead\_end\_stack[-1] == len(output\_track\_stack) + 1:  output\_track\_stack.append(dead\_end\_stack.pop())  # Проверяем, можно ли сразу отправить вагон на путь 2  if wagon == len(output\_track\_stack) + 1:  output\_track\_stack.append(wagon)  else:  # Если нельзя, отправляем вагон в тупик  dead\_end\_stack.append(wagon)  # Проверяем оставшиеся вагоны в тупике  while dead\_end\_stack and dead\_end\_stack[-1] == len(output\_track\_stack) + 1:  output\_track\_stack.append(dead\_end\_stack.pop())  # Если все вагоны успешно перемещены на путь 2, возвращаем True  return len(output\_track\_stack) == n  4  def main():  # Вводим количество вагонов  n: int = int(input())  # Вводим порядок вагонов на пути 1  wagons: List[int] = list(map(int, input().split()))  # Проверяем, можно ли отсортировать вагоны и выводим результат  print("YES" if can\_sort\_train(n, wagons) else "NO")  main() |

**Задание 7. Парикмахерская**

В парикмахерской работают три мастера. Каждый тратит на одного клиента ровно полчаса, а затем сразу переходит к следующему, если в очереди кто-то есть, либо ожидает, когда придет следующий клиент.

Даны времена прихода клиентов в парикмахерскую (в том порядке, в котором они приходили). Требуется для каждого клиента указать время, когда он выйдет из парикмахерской.

**Входные данные**

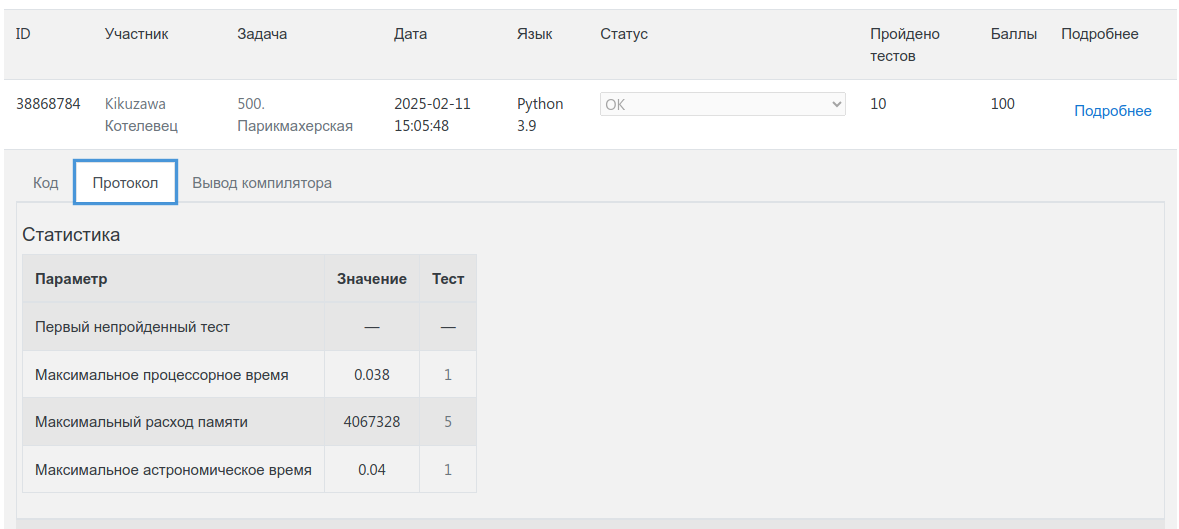
В первой строке вводится натуральное число *N*, не превышающее 100 – количество клиентов.

*N строках вводятся времена прихода клиентов – по два числа, обозначающие часы и минуты (часы – от 0 до 23, минуты – от 0 до 59). Времена указаны в порядке возрастания (все времена различны).*

*Гарантируется, что всех клиентов успеют обслужить до полуночи.*

***Выходные данные***

*Требуется вывести N пар чисел: времена выхода из парикмахерской 1-го, 2-го, …, N-го клиента (часы и минуты).*



Листинг программы

|  |
| --- |
| from typing import List, Tuple, cast, Iterable  from collections import deque  def calculate\_exit\_times(arrival\_times):  # Время окончания работы каждого мастера (в минутах с начала суток)  workers\_time: List[int] = [0, 0, 0]  # Очередь для хранения времени выхода клиентов (часы, минуты)  exit\_times: deque[Tuple[int, int]] = deque()  # Обрабатываем каждого клиента  for hours, minutes in arrival\_times:  # Переводим время прихода клиента в минуты с начала суток  arrival\_minutes = hours \* 60 + minutes  # Находим мастера, который освободится раньше всех  next\_worker\_index = workers\_time.index(min(workers\_time))  # Если мастер свободен к моменту прихода клиента, обслуживание начинается сразу  if workers\_time[next\_worker\_index] <= arrival\_minutes:  workers\_time[next\_worker\_index] = arrival\_minutes + 30  else:  # Если мастер занят, клиент ждет, пока мастер освободится  workers\_time[next\_worker\_index] += 30  # Вычисляем время выхода клиента и добавляем его в очередь  exit\_time = workers\_time[next\_worker\_index]  exit\_times.append((exit\_time // 60, exit\_time % 60)) # Переводим минуты обратно в часы и минуты  return exit\_times  def main():  # Вводим количество клиентов  n: int = int(input())  # Вводим времена прихода клиентов (часы, минуты)  arrival\_times: List[Tuple[int, int]] = cast(  List[Tuple[int, int]],  [tuple(map(int, input().split())) for \_ in range(n)]  )  # Вычисляем времена выхода клиентов  exit\_times: Iterable[Tuple[int, int]] = calculate\_exit\_times(arrival\_times)  # Выводим времена выхода для каждого клиента  for hours, minutes in exit\_times:  print(hours, minutes)  main() |

**Задание 8. Шифровка**

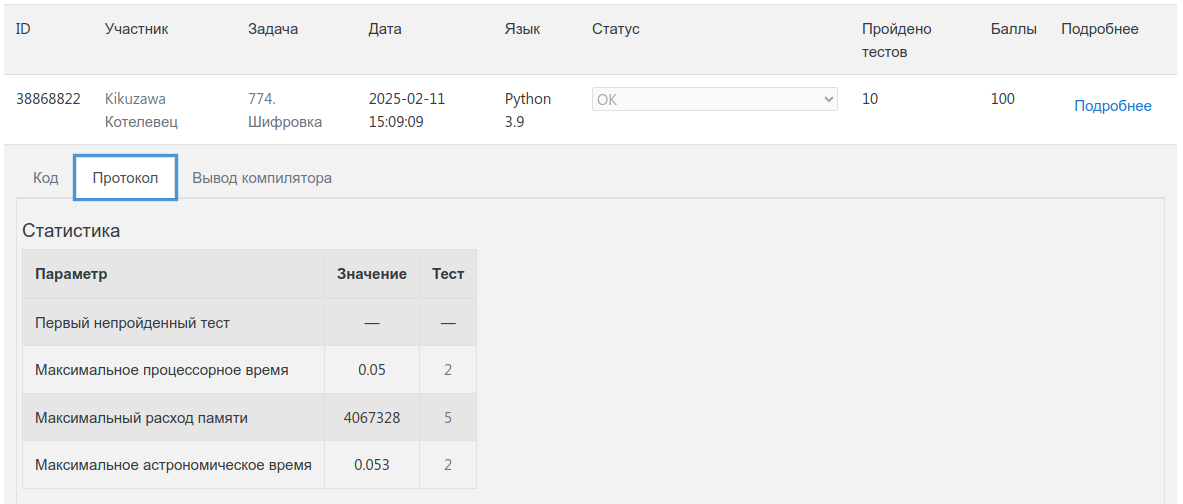
Шпион Коля зашифровал и послал в центр радиограмму. Он использовал такой способ шифровки: сначала выписал все символы своего сообщения (включая знаки препинания и т.п.), стоявшие на четных местах, в том же порядке, а затем ­– все символы, стоящие на нечетных местах. Напишите программу, которая расшифровывает сообщение.

**Входные данные**

Вводится одна непустая строка длиной не более 250 символов – зашифрованное сообщение. Строка может состоять из любых символов, кроме пробельных.

**Выходные данные**

Выведите одну строку – расшифрованное сообщение.



Листинг программы

|  |
| --- |
| from typing import List, AnyStr  def decrypt\_message(encrypted\_message):  """  Расшифровывает сообщение, которое было зашифровано методом Шпиона Коли.  Метод шифрования: сначала идут символы с чётными индексами,  затем символы с нечётными индексами.    Например: "Привет!" -> "Пиветр!"    Args:  encrypted\_message (AnyStr): Зашифрованное сообщение    Returns:  str: Расшифрованное исходное сообщение  """  # Преобразуем строку в список символов для возможности изменения элементов  symbols: List[AnyStr] = list(encrypted\_message)    # Находим середину строки - это длина первой части зашифрованного сообщения  n: int = len(encrypted\_message) // 2    # Переставляем символы на свои места  # Для каждой пары символов:  # - первый символ пары берём из второй половины (encrypted\_message[i + n])  # - второй символ пары берём из первой половины (encrypted\_message[i])  for i in range(n):  symbols[2 \* i: 2 \* i + 2] = encrypted\_message[i + n], encrypted\_message[i]    # Объединяем список символов обратно в строку  return ''.join(symbols)  def main():  """  Главная функция программы.  Читает зашифрованное сообщение и выводит его расшифрованную версию.  """  # Получаем зашифрованное сообщение от пользователя  encrypted\_message = input()    # Расшифровываем сообщение  decrypted\_message: str = decrypt\_message(encrypted\_message)    # Выводим результат  print(decrypted\_message)  # Запуск программы  main() |

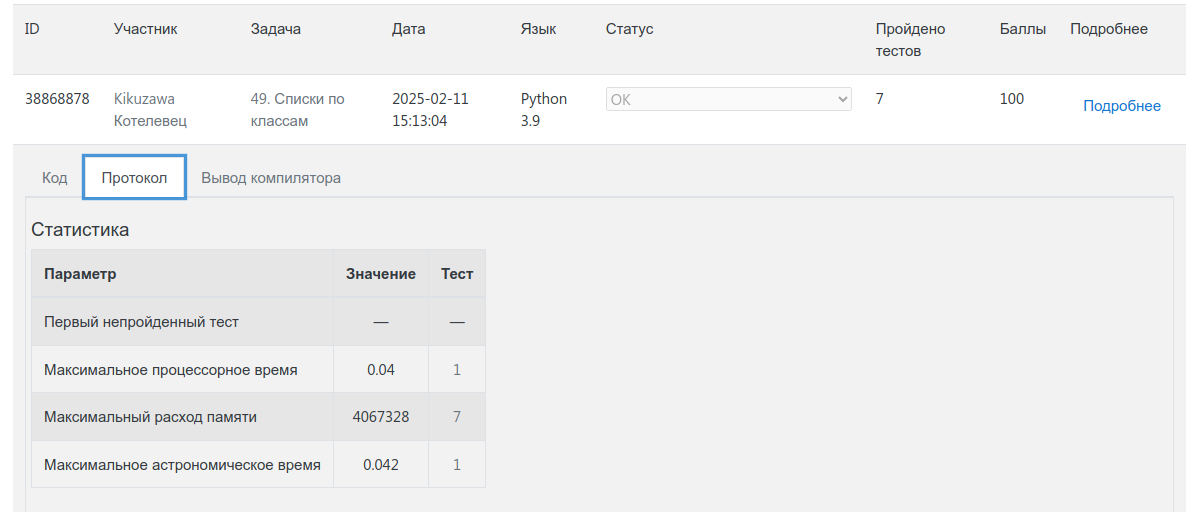
**Задание 9. Списки по классам**

***Формат входных данных***

В каждой строке сначала записан номер класса (число, равное 9, 10 или 11), затем (через пробел) – фамилия ученика. Общее число строк в файле не превосходит 100000. Длина каждой фамилии не превосходит 50 символов.

***Формат выходных данных***

Необходимо вывести список школьников по классам: сначала всех учеников 9 класса, затем – 10, затем – 11. Внутри одного класса порядок вывода фамилий должен быть таким же, как на входе.



Листинг программы

|  |
| --- |
| from typing import Dict, List, Iterable  def process\_students(students):  # Словарь для хранения учеников по классам  classes: Dict[int, List[str]] = {9: [], 10: [], 11: []}  # Обрабатываем каждого ученика  for student in students:  # Разделяем строку на номер класса и фамилию  class\_number, surname = student.split()  class\_number = int(class\_number) # Преобразуем номер класса в целое число  # Добавляем ученика в соответствующий класс  classes[class\_number].append(f"{class\_number} {surname}")  # Формируем итоговую строку для вывода  result = ""  # Проходим по классам в порядке 9, 10, 11  for class\_number in classes:  # Добавляем учеников текущего класса в результат  result += "\n".join(student for student in classes[class\_number]) + "\n"  return result  def main():  # Пути к входному и выходному файлам  input\_file\_path = "input.txt"  output\_file\_path = "output.txt"  # Чтение данных из входного файла  with open(input\_file\_path, 'r', encoding='KOI8-r') as file:  # Обрабатываем учеников и сортируем их по классам  students\_by\_class = process\_students(map(str.strip, file.readlines()))  # Запись результата в выходной файл  with open(output\_file\_path, 'w', encoding='KOI8-r') as file:  file.write(students\_by\_class)  main() |

**Задание 10. Игра в пьяницу**

В игре в пьяницу карточная колода раздается поровну двум игрокам. Далее они вскрывают по одной верхней карте, и тот, чья карта старше, забирает себе обе вскрытые карты, которые кладутся под низ его колоды. Тот, кто остается без карт – проигрывает.

Для простоты будем считать, что все карты различны по номиналу, а также, что самая младшая карта побеждает самую старшую карту ("шестерка берет туза").

Игрок, который забирает себе карты, сначала кладет под низ своей колоды карту первого игрока, затем карту второго игрока (то есть карта второго игрока оказывается внизу колоды).

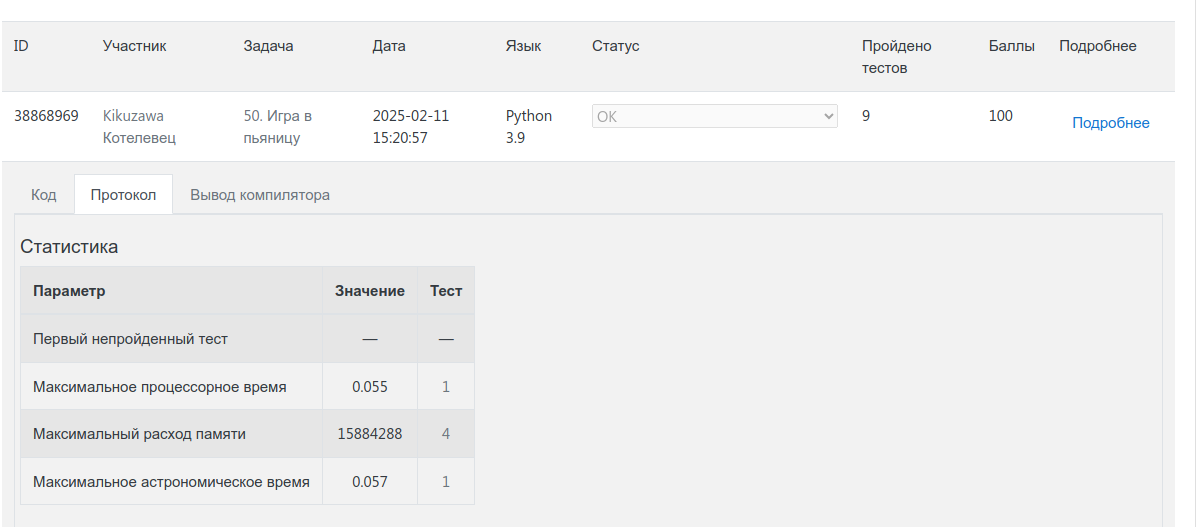
Напишите программу, которая моделирует игру в пьяницу и определяет, кто выигрывает. В игре участвует 10 карт, имеющих значения от 0 до 9, большая карта побеждает меньшую, карта со значением 0 побеждает карту 9.

**Входные данные**

Программа получает на вход две строки: первая строка содержит 5 чисел, разделенных пробелами — номера карт первого игрока, вторая – аналогично 5 карт второго игрока. Карты перечислены сверху вниз, то есть каждая строка начинается с той карты, которая будет открыта первой.

**Выходные данные**

Программа должна определить, кто выигрывает при данной раздаче, и вывести слово first или second, после чего вывести количество ходов, сделанных до выигрыша. Если на протяжении 106 ходов игра не заканчивается, программа должна вывести слово botva.



Листинг программы

|  |
| --- |
| from collections import deque  from typing import Tuple, Iterable, Dict  def is\_first\_player\_winner(card1, card2):  return (card1 > card2 and (card2, card1) != (0, 9)) or (card1 == 0 and card2 == 9)  def play\_game(  first\_deck: Iterable[int],  second\_deck: Iterable[int]  ) :    first\_deck: deque[int] = deque(first\_deck)  second\_deck: deque[int] = deque(second\_deck)  rounds: int = 0  max\_rounds: int = 1\_000\_000  while first\_deck and second\_deck:  rounds += 1  first\_card = first\_deck.popleft()  second\_card = second\_deck.popleft()  if is\_first\_player\_winner(first\_card, second\_card):  first\_deck.extend([first\_card, second\_card])  else:  second\_deck.extend([first\_card, second\_card])  if rounds >= max\_rounds:  return -1, -1  return not bool(first\_deck), rounds  def main():  cases: Dict[int, str] = {-1: "botva", 0: "first", 1: "second"}  result: Tuple[int, int] = play\_game(  map(int, input().split()),  map(int, input().split())  )  print(cases[result[0]], result[1] if result[1] != -1 else "")  main() |

Лабораторная работа 2

Задание 1.

При расследовании дорожно-транспортных происшествий часто возникают проблемы с розыском автомобилей, водители которых покинули место происшествия.

Получение свидетельских показаний – непростая работа. Ситуация осложняется тем, что очень часто свидетели могут только приблизительно вспомнить номер автомобиля. При этом с большой вероятностью опрашиваемый может перепутать порядок цифр или букв в номере.

По полученному от свидетеля происшествия номеру, подсчитайте, сколько различных номеров может получиться из него перестановкой букв и/или цифр, а также выведите все такие номера.

Напомним, что автомобильные номера в России состоят из трех букв и трех цифр, упорядоченных следующим образом: буква, три цифры, затем две буквы. Фрагмент номера, который идентифицирует регион, в котором зарегистрирован автомобиль, мы будем игнорировать.

В номере могут использоваться следующие буквы: «A», «B», «C», «E», «H», «K», «M», «O», «P», «T», «X», «Y» (эти буквы имеют схожие по написанию аналоги как в русском, так и в латинском алфавите). В этой задаче во входных данных будут использоваться буквы латинского алфавита.

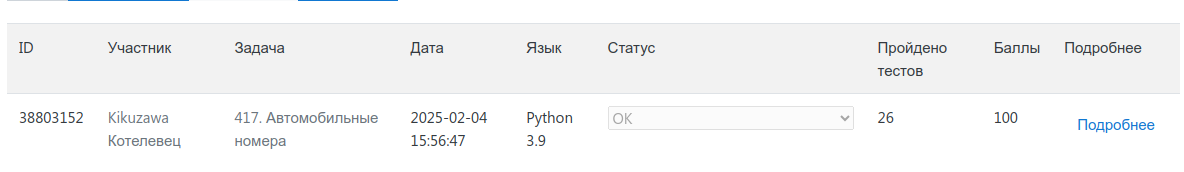
**Входные данные**

На вход программы поступает  одна строка, которая представляет собой корректный автомобильный номер.

**Выходные данные**

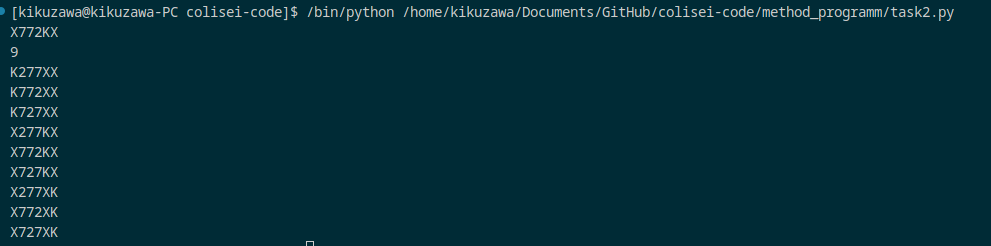
В первой строке  выведите число k – количество номеров, которые могут получиться из заданного перестановкой букв и/или цифр.

В последующих k строках выведите все такие номера в произвольном порядке.



Листинг программы:

|  |
| --- |
| from collections import deque  from itertools import permutations  from typing import Set, AnyStr, Sequence  **def** generate\_swaps(number: AnyStr) -> Sequence[AnyStr]:  letters: AnyStr = number[0] + number[4] + number[5]  digits: AnyStr = number[1] + number[2] + number[3]  unique\_digit\_swaps: Set[AnyStr] = {''.join(p) for p in permutations(digits)}  unique\_letter\_swaps: Set[AnyStr] = {''.join(p) for p in permutations(letters)}  all\_combinations: deque[AnyStr] = deque()  for letter in unique\_letter\_swaps:  for digit in unique\_digit\_swaps:  combined\_number = **f**"{letter[0]}{digit[0]}{digit[1]}{digit[2]}{letter[1]}{letter[2]}"  all\_combinations.append(combined\_number)  return all\_combinations  number\_by\_car: str = input()  swaps\_list: Sequence[AnyStr] = generate\_swaps(number\_by\_car)  print(len(swaps\_list), \*swaps\_list, sep='\n') |



Задача 2.

В классе учатся N человек. Классный руководитель получил указание направить на субботник R бригад по С человек в каждой.

Все бригады на субботнике будут заниматься переноской бревен. Каждое бревно одновременно несут все члены одной бригады. При этом бревно нести тем удобнее, чем менее различается рост членов этой бригады.

Числом неудобства бригады будем называть разность между ростом самого высокого и ростом самого низкого членов этой бригады (если в бригаде только один человек, то эта разница равна 0). Классный руководитель решил сформировать бригады так, чтобы максимальное из чисел неудобства сформированных бригад было минимально. Помогите ему в этом!

Рассмотрим следующий пример. Пусть в классе 8 человек, рост которых в сантиметрах равен 170, 205, 225, 190, 260, 130, 225, 160, и необходимо сформировать две бригады по три человека в каждой. Тогда одним из вариантов является такой:

1 бригада: люди с ростом 225, 205, 225

2 бригада: люди с ростом 160, 190, 170

При этом число неудобства первой бригады будет равно 20, а число неудобства второй — 30. Максимальное из чисел неудобств будет 30, и это будет наилучший возможный результат.

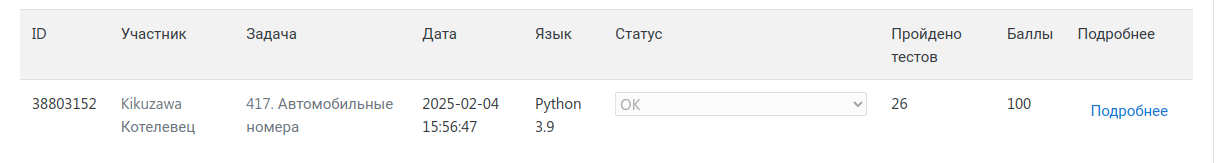
Формат входных данных

Сначала вводятся натуральные числа N, R и C — количество человек в классе, количество бригад и количество человек в каждой бригаде (1 ≤ R∙C ≤ N ≤ 100 000). Далее вводятся N целых чисел — рост каждого из N учеников. Рост ученика — натуральное число, не превышающее 1 000 000 000.

Формат выходных данных

Выведите одно число — наименьше возможное значение максимального числа неудобства сформированных бригад.

|  |
| --- |
| from typing import List, Iterable  def can\_form\_branches(  max\_uncomfortable: int,  discomforts: Iterable[int],  r: int,  c: int  ) -> bool:  count = 0  members\_in\_current\_branch = 0  for discomfort in discomforts:  members\_in\_current\_branch -= 1  if members\_in\_current\_branch < 1 and discomfort <= max\_uncomfortable:  count += 1  members\_in\_current\_branch = c  return count >= r  def find\_minimum\_max\_uncomfortable(n: int, r: int, c: int, heights: Iterable[int]) -> int:  heights: List[int] = sorted(heights)  discomforts: List[int] = [heights[i + c - 1] - heights[i] for i in range(n - c + 1)]  left = -1  right = heights[-1] - heights[0]  while left + 1 < right:  mid = (left + right) // 2  if can\_form\_branches(mid, discomforts, r, c):  right = mid  else:  left = mid  return right  def main() -> None:  n, r, c = map(int, input().split())  heights: List[int] = [int(input()) for \_ in range(n)]  result: int = find\_minimum\_max\_uncomfortable(n, r, c, heights)  print(result)  if \_\_name\_\_ == "\_\_main\_\_":  main() |



Задание 3.

Перед началом тараканьих бегов всем болельщикам было предложено сделать по две ставки на результаты бегов. Каждая ставка имеет вид "Таракан №A придет раньше, чем таракан №B".

Организаторы бегов решили выяснить, могут ли тараканы прийти в таком порядке, чтобы у каждого болельщика сыграла ровно одна ставка из двух (то есть чтобы ровно одно из двух утверждений каждого болельщика оказалось верным). Считается, что никакие два таракана не могут прийти к финишу одновременно.

Входные данные

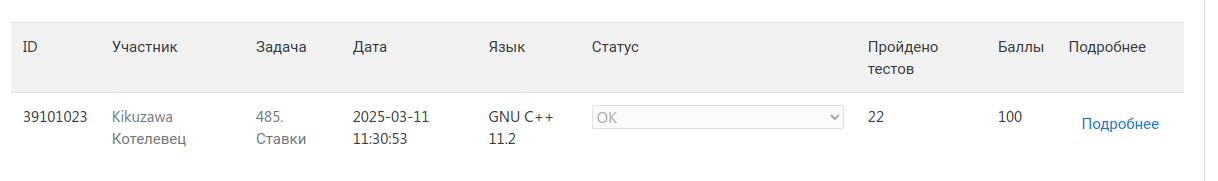
В первой строке входных данных содержатся два разделенных пробелом натуральных числа: число K, не превосходящее 10, - количество тараканов и число N, не превосходящее 100, - количество болельщиков. Все тараканы пронумерованы числами от 1 до K. Каждая из следующих N строк содержит 4 натуральных числа A, B, C, D, не превосходящих K, разделенных пробелами. Они соответствуют ставкам болельщика "Таракан №A придет раньше, чем таракан №B" и "Таракан №C придет раньше, чем таракан №D".

Выходные данные

Если завершить бега так, чтобы у каждого из болельщиков сыграла ровно одна из двух ставок, можно, то следует вывести номера тараканов в том порядке, в котором они окажутся в итоговой таблице результатов (сначала номер таракана, пришедшего первым, затем номер таракана, пришедшего вторым и т. д.) в одну строку через пробел. Если таких вариантов несколько, выведите любой из них.

Если требуемого результата добиться нельзя, выведите одно число 0.

|  |
| --- |
| #include <iostream>  #include <algorithm>  #include <vector>  using namespace std;  int main() {  int k, n;  cin >> k >> n;  vector<int> cockroach(k); // Перестановка тараканов  vector<vector<int>> Rates(n, vector<int>(4)); // Ставки болельщиков  // Считываем ставки  for (int i = 0; i < n; i++) {  cin >> Rates[i][0] >> Rates[i][1] >> Rates[i][2] >> Rates[i][3];  }  // Инициализация первой перестановки  for (int i = 0; i < k; i++) {  cockroach[i] = i + 1;  }  // Перебор всех перестановок  do {  bool valid = true; // Флаг для проверки условий ставок  // Создаем массив позиций для быстрого доступа  vector<int> positions(k + 1);  for (int i = 0; i < k; i++) {  positions[cockroach[i]] = i;  }  // Проверяем каждую ставку  for (int i = 0; i < n; i++) {  int a = Rates[i][0], b = Rates[i][1], c = Rates[i][2], d = Rates[i][3];  // Индексы тараканов  int index\_a = positions[a];  int index\_b = positions[b];  int index\_c = positions[c];  int index\_d = positions[d];  // Условия ставок  bool condition1 = (index\_a < index\_b); // A до B  bool condition2 = (index\_c < index\_d); // C до D  // Проверяем, чтобы ровно одно из условий было выполнено  if ((condition1 && condition2) || (!condition1 && !condition2)) {  valid = false;  break;  }  }  // Если все ставки выполнены, выводим результат  if (valid) {  for (int i = 0; i < k; i++) {  cout << cockroach[i] << " ";  }  cout << endl;  return 0;  }  } while (next\_permutation(cockroach.begin(), cockroach.end()));  // Если не нашли подходящей перестановки  cout << 0 << endl;  return 0;  } |



Задание 4.

В турнире по хоккею участвовало K команд, каждая сыграла с каждой по одному матчу. За победу команда получала 2 очка, за ничью – 1, за поражение – 0 очков.

Известно, сколько очков в итоге получила каждая команда, однако результаты конкретных матчей были утеряны. Требуется восстановить одну из возможных турнирных таблиц.

Входные данные

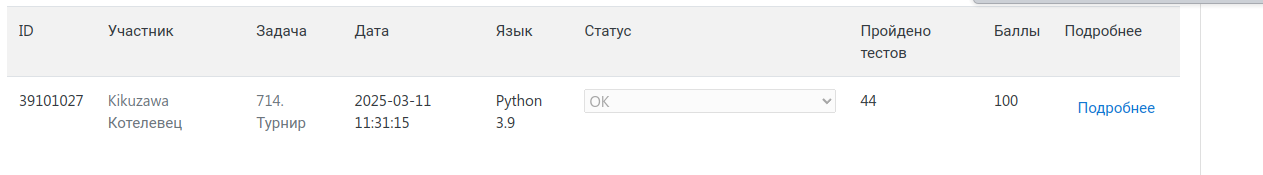
В первой строке входных данных содержится одно натурально число K, не превосходящее 100 – количество команд. Во второй строке  задаются  через пробел K целых неотрицательных чисел, не превосходящих 2(K–1), – количество очков, набранных командами, занявшими первое, второе, …, K-е места соответственно (то есть каждое следующее число не больше предыдущего).

Выходные данные

Выведите турнирную таблицу в следующем формате. Таблица должна состоять из K строк с результатами игр команд, занявших первое, второе, …, последнее место (команды, набравшие одинаковое число очков, могут быть расположены в таблице в любом порядке). В каждой строке должно быть записано K чисел через пробел – количество очков, набранных в игре данной команды с первой, второй, … командами соответственно. Количество очков – это число 0, 1 или 2. В клетках на главной диагонали (соответствующих не существующей игре команды "самой с собой") нужно записать нули.

Гарантируется, что входные данные соответствуют реальному турниру, то есть хотя бы одна таблица, соответствующая входным данным, может быть построена. Если таких таблиц несколько, выведите любую из них.

|  |
| --- |
| from typing import List, Iterable  def restore\_tournament\_table(num\_teams: int, points: List[int]) -> Iterable[Iterable[int]]:  # Инициализация таблицы результатов  tournament\_table: List[List[int]] = [[0] \* num\_teams for \_ in range(num\_teams)]  # Индексы команд  team\_indices = range(num\_teams)  for i in range(num\_teams - 1):  # Сортируем команды по очкам  sorted\_points, sorted\_indices = zip(\*sorted(zip(points, team\_indices)))  sorted\_points: List[int] = list(sorted\_points)  sorted\_indices: List[int] = list(sorted\_indices)  for j in range(i + 1, num\_teams):  if tournament\_table[sorted\_indices[i]][sorted\_indices[j]] > 0:  continue  elif sorted\_points[i] > 0:  # Ничья  tournament\_table[sorted\_indices[i]][sorted\_indices[j]] = 1  tournament\_table[sorted\_indices[j]][sorted\_indices[i]] = 1  sorted\_points[i] -= 1  points[sorted\_indices[j]] -= 1  elif sorted\_points[i] == 0:  # Поражение  tournament\_table[sorted\_indices[j]][sorted\_indices[i]] = 2  points[sorted\_indices[j]] -= 2  points[sorted\_indices[i]] = -i - 1  return tournament\_table  def main() -> None:  num\_teams: int = int(input())  points: List[int] = list(map(int, input().split()))  tournament\_table: Iterable[Iterable[int]] = restore\_tournament\_table(num\_teams, points)  for row in tournament\_table:  print(\*row)  if \_\_name\_\_ == "\_\_main\_\_":  main() |



Задание 5.

Дано два списка чисел, числа в первом списке упорядочены по неубыванию. Для каждого числа из второго списка определите номер первого и последнего появления этого числа в первом списке.

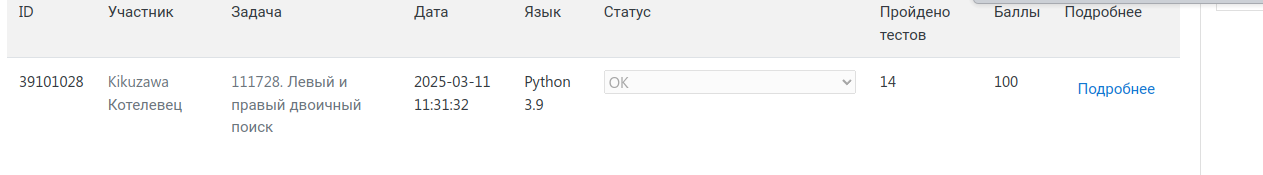
Входные данные

В первой строке входных данных записано два числа NN и MM (1≤N,M≤200001≤N,M≤20000). Во второй строке записано NN упорядоченных по неубыванию целых чисел — элементы первого списка. В третьей строке записаны MM целых неотрицательных чисел - элементы второго списка. Все числа в списках - целые 32-битные знаковые.

Выходные данные

Программа должна вывести MM строчек. Для каждого числа из второго списка нужно вывести номер его первого и последнего вхождения в первый список. Нумерация начинается с единицы. Если число не входит в первый список, нужно вывести одно число 00.

|  |
| --- |
| import bisect  from typing import List, Tuple, Sequence, Union  from collections import deque  def find\_first\_and\_last\_occurrences(  sorted\_list: Sequence[int],  search\_list: Sequence[int]  ) -> deque[Union[Tuple[int, int], Tuple[int]]]:  results: deque[Union[Tuple[int, int], Tuple[int]]] = deque()  for number in search\_list:  first\_index = bisect.bisect\_left(sorted\_list, number)  if first\_index >= len(sorted\_list) or sorted\_list[first\_index] != number:  results.append((0,))  else:  last\_index = bisect.bisect\_right(sorted\_list, number) - 1  results.append((first\_index + 1, last\_index + 1)) # +1 для единичной нумерации  return results  def main() -> None:  \_, \_ = map(int, input().split())  sorted\_list: List[int] = list(map(int, input().split()))  search\_list: List[int] = list(map(int, input().split()))  occurrences = find\_first\_and\_last\_occurrences(sorted\_list, search\_list)  for result in occurrences:  print(\*result)  if \_\_name\_\_ == "\_\_main\_\_":  main() |



Задание 6.

Дано N

натуральных чисел. Требуется для каждого числа найти количество вариантов разбиения его на сумму двух других чисел из данного набора.

Входные данные

В первой строке дано число N

( 1 ≤ N

≤ 10000). Далее заданы N

натуральных чисел, не превосходящих 109

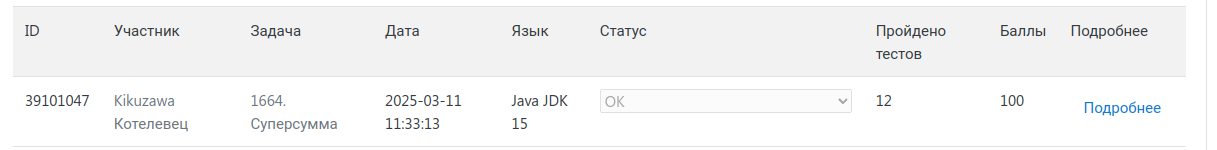
. Для каждого числа количество разбиений меньше 231.

Выходные данные

Вывести N

чисел – количество разбиений, в порядке, соответствующем исходному.

|  |
| --- |
| import java.util.Arrays;  import java.util.HashMap;  import java.util.Map;  import java.util.Scanner;  import java.util.stream.Collectors;  public class SuperSum {  static int[] countPartitions(int[] numbers) {  Map<Integer, Long> countMap = Arrays.stream(numbers)  .boxed()  .collect(Collectors.groupingBy(n -> n, Collectors.counting()));  int[] results = new int[numbers.length];  for (var i = 0; i < numbers.length; i++) {  int number = numbers[i];  int count = 0;  for (Map.Entry<Integer, Long> entry : countMap.entrySet()) {  var x = entry.getKey();  var y = number - x;  // Избегаем дублирования  if (y < x) {  continue;  }  var xCount = entry.getValue();  if (x == y) {  // Если x и y одинаковые, то выбираем 2 из countMap[x]  count += xCount \* (xCount - 1) / 2;  } else if (countMap.containsKey(y)) {  // Если x и y разные, то просто перемножаем их количества  count += xCount \* countMap.get(y);  }  }  results[i] = count;  }  return results;  }  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  int n = scanner.nextInt();  int[] numbers = new int[n];  for (int i = 0; i < n; i++) {  numbers[i] = scanner.nextInt();  }  int[] partitions = countPartitions(numbers);  for (int result : partitions) {  System.out.println(result);  }  scanner.close();  }  } |



Задание 7.

На прямой расположены стойла, в которые необходимо расставить коров так, чтобы минимальное расcтояние между коровами было как можно больше.

Входные данные

В первой строке вводятся числа N

(2<N<10001)

– количество стойл и K

(1<K<N)

– количество коров. Во второй строке задаются N

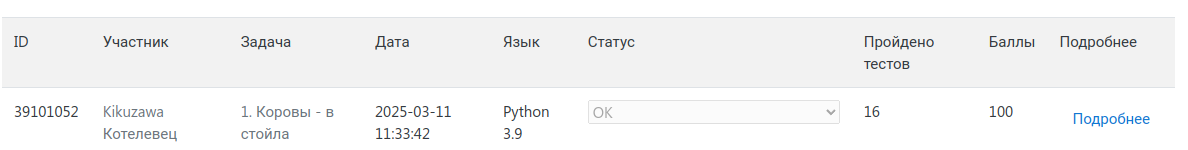
натуральных чисел в порядке возрастания – координаты стойл (координаты не превосходят 109

)

Выходные данные

Выведите одно число – наибольшее возможное допустимое расстояние.

|  |
| --- |
| from typing import List, Sequence  def max\_distance\_between\_cows(stalls: Sequence[int], num\_cows: int) -> int:  left: int = 0  right: int = stalls[-1] - stalls[0] + 1  while left < right:  mid: int = (left + right) // 2  cows\_placed: int = 1  last\_placed\_stall: int = stalls[0] # Последнее стойло, в которое была поставлена корова  for current\_stall in stalls[1:]:  if current\_stall - last\_placed\_stall >= mid:  cows\_placed += 1  last\_placed\_stall = current\_stall  if cows\_placed >= num\_cows:  left = mid + 1  else:  right = mid  return left - 1  def main() -> None:  n, k = map(int, input().split())  stalls: List[int] = list(map(int, input().split()))  result: int = max\_distance\_between\_cows(stalls, k)  print(result)  if \_\_name\_\_ == "\_\_main\_\_":  main() |



Задание 8.

Дано N отрезков провода длиной L1, L2, ..., LN сантиметров. Требуется с помощью разрезания получить из них K равных отрезков как можно большей длины, выражающейся целым числом сантиметров. Если нельзя получить K отрезков длиной даже 1 см, вывести 0.

Ограничения: 1 <= N <= 10 000, 1 <= K <= 10 000, 100 <= Li <= 10 000 000, все числа целые.

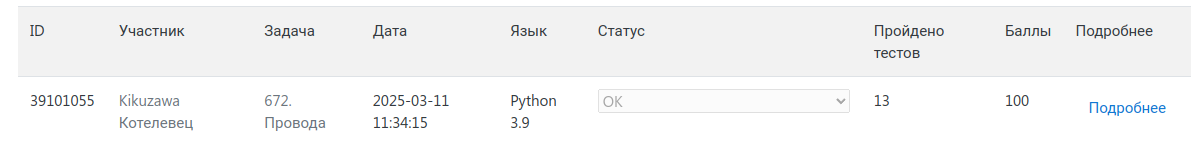
Входные данные

В первой строке находятся числа N и К. В следующих N строках - L1, L2, ..., LN, по одному числу в строке.

Выходные данные

Вывести одно число - полученную длину отрезков.

|  |
| --- |
| from typing import List, Sequence  def max\_segment\_length(segments: Sequence[int], required\_segments: int) -> int:  left: int = 1  right: int = 10\_000\_001  while left < right:  mid: int = (left + right) // 2  total\_segments: int = sum(length // mid for length in segments)  if total\_segments >= required\_segments:  left = mid + 1  else:  right = mid  return left - 1  def main() -> None:  n, k = map(int, input().split())  segments: List[int] = [int(input()) for \_ in range(n)]  result: int = max\_segment\_length(segments, k)  print(result)  if \_\_name\_\_ == "\_\_main\_\_":  main() |



Задание 9.

Банк «Кисловодск» переходит на новый вид банковских карт. Для этого производятся одинаковые заготовки, на которых есть специальное место для идентификации клиента. Изначально на этом месте записывается кодовое число X. В банке с помощью специального прибора можно стирать некоторые цифры числа X. Оставшиеся цифры, будучи записанными подряд, должны образовывать номер счета клиента. Например, при X = 12013456789 номера счетов 5, 12, 17 или 12013456789 получить можно, а номера 22 или 71 получить нельзя.

Способ распределения номеров счетов в банке очень прост. Счетам присваиваются последовательно номера 1, 2, … Очевидно, что при таком способе в какой-то момент впервые найдется номер счета N, который нельзя будет получить из цифр X указанным выше способом. Руководство банка хочет знать значение N.

Напишите программу, которая находила бы N по заданному X.

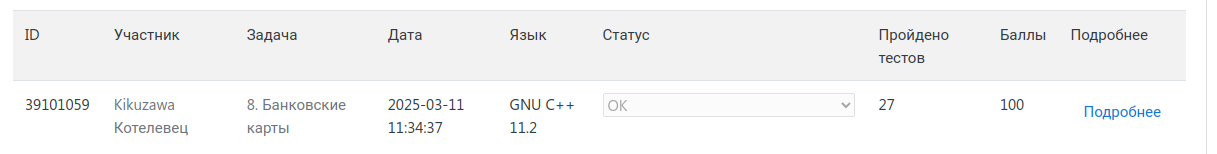
Входные данные

Вводится натуральное число X без ведущих нулей (1 ≤ X < 101000)

Выходные данные

Выведите искомое N без ведущих нулей.

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  //#define int int64\_t  void solve() {  string s;  cin >> s;  int n = int(s.size());  vector<int> x(n);  for (int i = 0; i < n; i++) {  x[i] = int(s[i] - '0');  }  vector<vector<int>> blocks;  int cur\_block = 0;  blocks.push\_back({});  vector<bool> cnt(10, false);  for (int i = n - 1; i >= 0; i--) {  cnt[x[i]] = true;  blocks[cur\_block].push\_back(x[i]);  bool check\_all = true;  for (int j = 0; j < 10; j++) {  if (cnt[j] == false) {  check\_all = false;  break;  }  }  if (check\_all) {  cur\_block++;  blocks.push\_back({});  for (int j = 0; j < 10; j++) {  cnt[j] = false;  }  }  }  for (int i = 0; i <= cur\_block; i++) {  reverse(blocks[i].begin(), blocks[i].end());  }  /\*for (auto to : blocks) {  for (auto to2 : to) {  cout << to2;  }  cout << '\n';  }\*/  for (int j = 0; j < 10; j++) {  cnt[j] = false;  }  for (auto num : blocks[cur\_block]) {  cnt[num] = true;  }  int first\_digit = 0;  for (int j = 1; j < 10; j++) {  if (cnt[j] == false) {  first\_digit = j;  break;  }  }  if (first\_digit == 0) {  cout << 10;  }  else {  cout << first\_digit;  }  int cur\_digit = first\_digit;  for (int i = cur\_block - 1; i >= 0; i--) {  for (int j = 0; j < 10; j++) {  cnt[j] = false;  }  bool ff = true;  for (auto digit : blocks[i]) {  if (ff == false) {  cnt[digit] = true;  }  if (digit == cur\_digit && ff == true) {  ff = false;  }  }  for (int j = 0; j < 10; j++) {  if (cnt[j] == false) {  cout << j;  cur\_digit = j;  break;  }  }  }  }  int32\_t main() {  ios\_base::sync\_with\_stdio(0);  cin.tie(0);  cout.tie(0);  //freopen("inputik.txt", "r", stdin);  //freopen("outputik.txt", "w", stdout);  int t = 1;  //cin >> t;  while (t--) {  solve();  }  } |



Задание 10

Глеб обожает шоппинг. Как-то раз он загорелся идеей подобрать себе кепку, майку, штаны и ботинки так, чтобы выглядеть в них максимально стильно. В понимании Глеба стильность одежды тем больше, чем меньше разница в цвете элементов его одежды.

В наличии имеется N1 кепок, N2 маек, N3 штанов и N4 пар ботинок (1 ≤ Ni ≤ 100 000). Про каждый элемент одежды известен его цвет (целое число от 1 до 100 000). Комплект одежды — это одна кепка, майка, штаны и одна пара ботинок. Каждый комплект характеризуется максимальной разницей между любыми двумя его элементами. Помогите Глебу выбрать максимально стильный комплект, то есть комплект с минимальной разницей цветов.

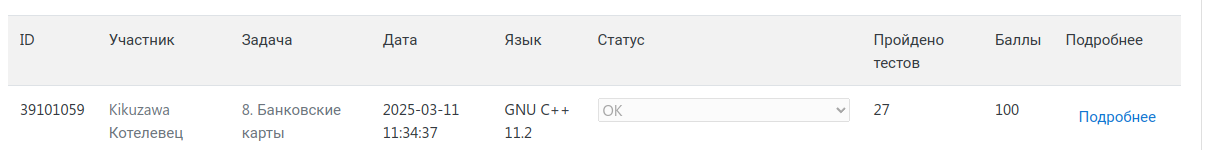
Входные данные

Для каждого типа одежды i (i = 1, 2, 3, 4) сначала вводится количество Ni элементов одежды этого типа, далее в следующей строке — последовательность из Ni целых чисел, описывающих цвета элементов. Все четыре типа подаются на вход последовательно, начиная с кепок и заканчивая ботинками. Все вводимые числа целые, положительные и не превосходят 100 000.

Выходные данные

Выведите четыре целых числа — цвета соответственно для кепки, майки, штанов и ботинок, которые должен выбрать Глеб из имеющихся для того, чтобы выглядеть наиболее стильно. Если ответов несколько, выведите любой.

|  |
| --- |
| from array import array, ArrayType  from copy import copy  from typing import List, Tuple, Sequence  def find\_most\_stylish\_outfit(  hats: Sequence[int],  shirts: Sequence[int],  pants: Sequence[int],  shoes: Sequence[int]  ) -> Tuple[int, int, int, int]:  indices: ArrayType[int] = array('i', [0, 0, 0, 0])  min\_diff: float = float('inf')  best\_indices: ArrayType[int] = array('i', [0, 0, 0, 0])  while all(indices[i] < len(lst) for i, lst in enumerate((hats, shirts, pants, shoes))):  current\_colors = (hats[indices[0]], shirts[indices[1]], pants[indices[2]], shoes[indices[3]])  min\_color = min(current\_colors)  max\_color = max(current\_colors)  current\_diff = max\_color - min\_color  # Обновляем минимальную разницу и сохраняем текущие индексы  if current\_diff < min\_diff:  min\_diff = current\_diff  best\_indices = copy(indices)  # Если разница минимальна (0), выходим из цикла  if current\_diff == 0:  break  # Сдвигаем указатель у минимального элемента  for i, color in enumerate(current\_colors):  if color == min\_color:  indices[i] += 1  break  return (  hats[best\_indices[0]],  shirts[best\_indices[1]],  pants[best\_indices[2]],  shoes[best\_indices[3]]  )  def main() -> None:  colors\_for\_clothes: List[List[int]] = []  for \_ in range(4):  \_ = int(input())  colors = sorted(map(int, input().split()))  colors\_for\_clothes.append(colors)  stylish\_outfit: Tuple[int, int, int, int] = find\_most\_stylish\_outfit(\*colors\_for\_clothes)  print(\*stylish\_outfit)  if \_\_name\_\_ == "\_\_main\_\_":  main() |



Задание 11.

Дано кубическое уравнение ax3+bx2+cx+d=0(a≠0)ax3+bx2+cx+d=0(a≠0). Известно, что у этого уравнения ровно один корень. Требуется его найти.

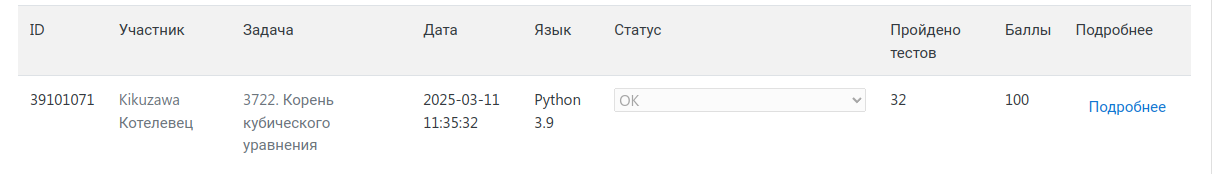
Входные данные

Во входных данных через пробел записаны четыре целых числа: −1000≤a,b,c,d≤1000−1000≤a,b,c,d≤1000.

Выходные данные

Выведите единственный корень уравнения с точностью не менее 4 знаков после десятичной точки.

|  |
| --- |
| def evaluate\_polynomial(a: int, b: int, c: int, d: int, x: float) -> float:  """Вычисляет значение кубического уравнения ax^3 + bx^2 + cx + d."""  return a \* x \*\* 3 + b \* x \*\* 2 + c \* x + d  def evaluate\_derivative(a: int, b: int, c: int, x: float) -> float:  """Вычисляет значение производной кубического уравнения 3ax^2 + 2bx + c."""  return 3 \* a \* x \*\* 2 + 2 \* b \* x + c  def find\_root\_newton(a: int, b: int, c: int, d: int, initial\_guess: float = 0.0, tolerance: float = 1e-5) -> float:  """Находит корень кубического уравнения методом Ньютона."""  x0 = initial\_guess  while True:  fx = evaluate\_polynomial(a, b, c, d, x0)  f\_prime\_x = evaluate\_derivative(a, b, c, x0)  # Новое приближение корня  x1 = x0 - fx / f\_prime\_x  # Если разница между приближениями достаточно мала, завершаем итерацию  if abs(x1 - x0) < tolerance:  return x1 # Возвращаем найденный корень  # Обновляем приближение  x0 = x1  def bisection(a: int, b: int, c: int, d: int, left: float, right: float, tol: float = 1e-5) -> float:  """Находит корень кубического уравнения методом бисекции."""  if evaluate\_polynomial(a, b, c, d, left) \* evaluate\_polynomial(a, b, c, d, right) >= 0:  return find\_root\_newton(a, b, c, d) # Если нет корня, используем метод Ньютона  while (right - left) / 2 > tol:  midpoint = (left + right) / 2  f\_mid = evaluate\_polynomial(a, b, c, d, midpoint)  if f\_mid == 0:  return midpoint # Найден точный корень  elif evaluate\_polynomial(a, b, c, d, left) \* f\_mid < 0:  right = midpoint  else:  left = midpoint  return (left + right) / 2 # Возвращаем приближенный корень  def main() -> None:  """Основная функция для чтения входных данных и вывода корня уравнения."""  a, b, c, d = map(int, input().split())  root = bisection(a, b, c, d, -1000, 1000)  print(root)  if \_\_name\_\_ == "\_\_main\_\_":  main() |



Задание 12.

Студенты одного из вузов спроектировали робота для частичной автоматизации процесса сборки авиационного двигателя.

В процессе сборки двигателя могут встречаться операции 26 типов, которые обозначаются строчными буквами латинского алфавита. Процесс сборки состоит из N операций.

Предполагается использовать робота один раз для выполнения части подряд идущих операций из процесса сборки.

Память робота состоит из K ячеек, каждая из которых содержит одну операцию. Операции выполняются последовательно, начиная с первой, в том порядке, в котором они расположены в памяти. Выполнив последнюю из них, робот продолжает работу с первой. Робота можно остановить после любой операции. Использование робота экономически целесообразно, если он выполнит хотя бы K + 1 операцию.

Требуется написать программу, которая по заданному процессу сборки определит количество экономически целесообразных способов использования робота.

Входные данные

В первой строке входного файла записано число K > 0 "— количество операций, которые можно записать в память робота.

Вторая строка состоит из N > K строчных латинских букв, обозначающих операции "— процесс сборки двигателя. Операции одного и того же типа обозначаются одной и той же буквой.

Выходные данные

Выходной файл должен содержать единственное целое число "— количество экономически целесообразных способов использования робота.

Примечание

Данная задача содержит четыре подзадачи. Для оценки каждой подзадачи используется своя группа тестов. Баллы за подзадачу начисляются только в том случае, если все тесты из этой группы пройдены.

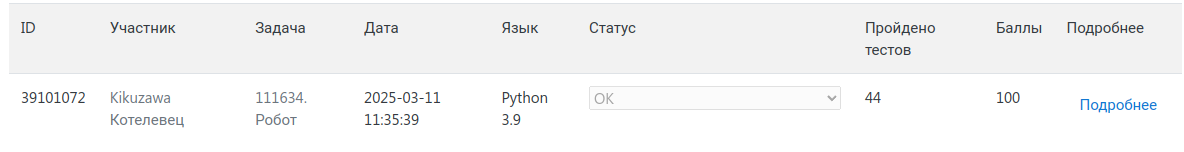
Тесты из условия. Подзадача оценивается в 0 баллов.

N ≤ 100. Подзадача оценивается в 30 баллов.

N ≤ 2000. Подзадача оценивается в 30 баллов.

N ≤ 200 000. Подзадача оценивается в 40 баллов.

|  |
| --- |
| def count\_economically\_viable\_ways(k: int, operations: str) -> int:  n: int = len(operations)  viable\_ways: int = 0  consecutive\_matches: int = 0  # Проходим по строке операций с конца к началу  for i in range(n - k - 1, -1, -1):  if operations[i] == operations[i + k]:  consecutive\_matches += 1  else:  consecutive\_matches = 0  viable\_ways += consecutive\_matches  return viable\_ways  def main() -> None:  k: int = int(input())  operations: str = input()  result: int = count\_economically\_viable\_ways(k, operations)  print(result)  if \_\_name\_\_ == "\_\_main\_\_":  main() |



Лабораторная работа 3

Задание 1.

Даны две последовательности, требуется найти и вывести их наибольшую общую подпоследовательность.

Входные данные

В первой строке входных данных содержится число N

– длина первой последовательности (1 ≤ N

≤ 1000). Во второй строке заданы члены первой последовательности (через пробел) – целые числа, не превосходящие 10000 по модулю.

В третьей строке записано число M

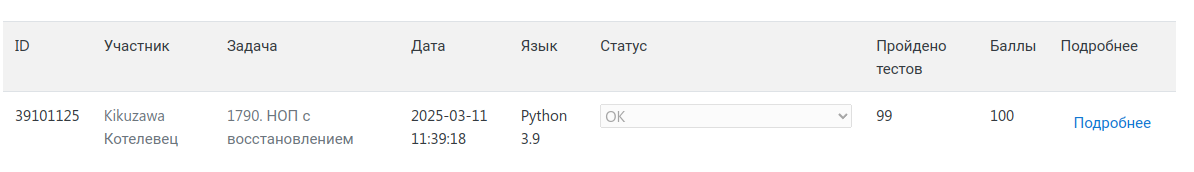
– длина второй последовательности (1 ≤ M

≤ 1000). В четвертой строке задаются члены второй последовательности (через пробел) – целые числа, не превосходящие 10000 по модулю.

Выходные данные

Требуется вывести наибольшую общую подпоследовательность данных последовательностей, через пробел.

|  |
| --- |
| from typing import List, Sequence  from itertools import product  def find\_longest\_common\_subsequence(  len\_seq\_a: int,  sequence\_a: Sequence[int],  len\_seq\_b: int,  sequence\_b: Sequence[int]  ) -> List[int]:  # Создаем таблицу для хранения длины LCS  lcs\_length\_table: List[List[int]] = [[0] \* (len\_seq\_b + 1) for \_ in range(len\_seq\_a + 1)]  # Заполняем таблицу  for i, j in product(range(1, len\_seq\_a + 1), range(1, len\_seq\_b + 1)):  if sequence\_a[i - 1] == sequence\_b[j - 1]:  lcs\_length\_table[i][j] = lcs\_length\_table[i - 1][j - 1] + 1  else:  lcs\_length\_table[i][j] = max(lcs\_length\_table[i - 1][j], lcs\_length\_table[i][j - 1])  # Восстанавливаем LCS  longest\_common\_subseq: List[int] = []  i: int = len\_seq\_a  j: int = len\_seq\_b  while i > 0 and j > 0:  if sequence\_a[i - 1] == sequence\_b[j - 1]:  longest\_common\_subseq.append(sequence\_a[i - 1])  i -= 1  j -= 1  elif lcs\_length\_table[i - 1][j] == lcs\_length\_table[i][j]:  i -= 1  else:  j -= 1  return longest\_common\_subseq[::-1]  def main() -> None:  len\_seq\_a: int = int(input())  sequence\_a: List[int] = list(map(int, input().split()))  len\_seq\_b: int = int(input())  sequence\_b: List[int] = list(map(int, input().split()))  longest\_common\_subseq: List[int] = find\_longest\_common\_subsequence(len\_seq\_a, sequence\_a, len\_seq\_b, sequence\_b)  print(' '.join(map(str, longest\_common\_subseq)))  if \_\_name\_\_ == "\_\_main\_\_":  main() |



Задание 2.

Дана текстовая строка. С ней можно выполнять следующие операции:

1. Заменить один символ строки на другой символ.

2. Удалить один произвольный символ.

3. Вставить произвольный символ в произвольное место строки.

Например, при помощи первой операции из строки "СОК" можно получить строку "СУК", при помощи второй операции - строку "ОК", при помощи третьей операции - строку "СТОК.

Минимальное количество таких операций, при помощи которых можно из одной строки получить другую, называется стоимостью редактирования или расстоянием Левенштейна.

Определите расстояние Левенштейна для двух данных строк.

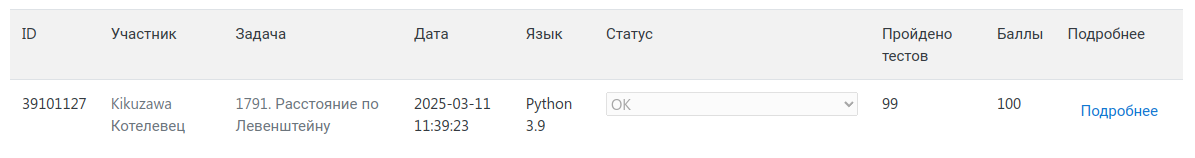
Входные данные

Программа получает на вход две строки, длина каждой из которых не превосходит 1000 символов, строки состоят только из заглавных латинских букв.

Выходные данные

Требуется вывести одно число – расстояние Левенштейна для данных строк.

|  |
| --- |
| from typing import List, AnyStr  from itertools import product  def levenshtein\_distance(source: AnyStr, target: AnyStr) -> int:  len\_source: int = len(source)  len\_target: int = len(target)  # Создаем таблицу для хранения расстояний  distance\_table: List[List[int]] = [[0] \* (len\_target + 1) for \_ in range(len\_source + 1)]  # Инициализация первой строки и первого столбца  for i, j in zip(range(len\_source + 1), range(len\_target + 1)):  distance\_table[i][0] = i # Расстояние от source до пустой строки  distance\_table[0][j] = j # Расстояние от пустой строки до target  # Заполняем таблицу  for i, j in product(range(1, len\_source + 1), range(1, len\_target + 1)):  if source[i - 1] == target[j - 1]:  distance\_table[i][j] = distance\_table[i - 1][j - 1] # Символы совпадают  else:  distance\_table[i][j] = 1 + min(distance\_table[i - 1][j],  distance\_table[i][j - 1],  distance\_table[i - 1][j - 1]) # Удаление, вставка, замена  return distance\_table[len\_source][len\_target]  def main() -> None:  a: str = input()  b: str = input()  distance = levenshtein\_distance(a, b)  print(distance)  if \_\_name\_\_ == "\_\_main\_\_":  main() |



Задание 3.

Числовая последовательность задана рекуррентной формулой: ai+1ai+1=(k∗aik∗ai+bb)mod mm. Найдите её наибольшую возрастающую подпоследовательность.

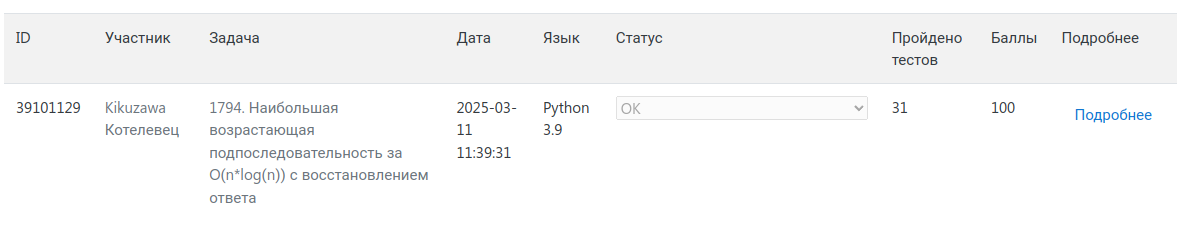
Входные данные

Программа получает на вход пять целых чисел: длину последовательности nn (1≤nn≤105105), начальный элемент последовательности a1a1, параметры kk, bb, mm для вычисления последующих членов последовательности (1≤mm≤104104, 0≤kk<mm, 0≤bb<mm, 0≤a1a1<mm).

Выходные данные

Требуется вывести наибольшую возрастающую подпоследовательность данной последовательности, разделяя числа пробелами. Если таких последовательностей несколько, необходимо вывести одну (любую) из них.

|  |
| --- |
| from bisect import bisect\_left  from typing import List, Sequence, Iterable  from collections import deque  def longest\_increasing\_subsequence(sequence: Sequence[int]) -> Iterable[int]:  largest\_increasing\_subsequence: List[int] = []  prev\_index: List[int] = [-1] \* len(sequence)  indices: List[int] = []  for i, value in enumerate(sequence):  pos = bisect\_left(largest\_increasing\_subsequence, value)  # Если значение больше всех элементов в lis, добавляем его  if pos == len(largest\_increasing\_subsequence):  largest\_increasing\_subsequence.append(value)  indices.append(i)  else:  largest\_increasing\_subsequence[pos] = value  indices[pos] = i  # Восстанавливаем индексы  if pos > 0:  prev\_index[i] = indices[pos - 1]  # Восстанавливаем саму последовательность  result: deque[int] = deque()  k = indices[-1]  while k != -1:  result.append(sequence[k])  k = prev\_index[k]  result.reverse()  return result  def main() -> None:  n, a1, k, b, m = map(int, input().split())  # Генерируем последовательность по формуле, которая дана в условии  sequence = [0] \* n  sequence[0] = a1  for i in range(1, n):  sequence[i] = (k \* sequence[i - 1] + b) % m  lis = longest\_increasing\_subsequence(sequence)  print(" ".join(map(str, lis)))  if \_\_name\_\_ == "\_\_main\_\_":  main() |



Задание 4.

Дана последовательность, требуется найти её наибольшую возрастающую подпоследовательность.

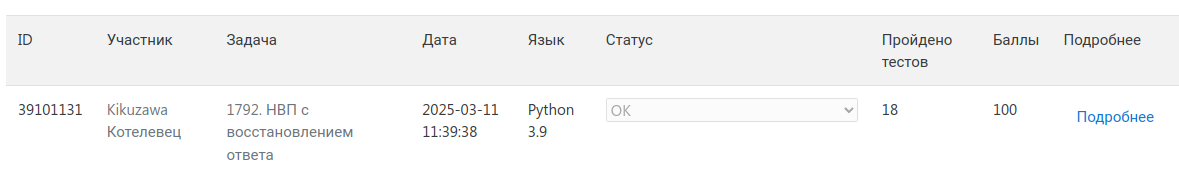
Входные данные

В первой строке входных данных задано число NN - длина последовательности (1 ≤ NN ≤ 1000). Во второй строке задается сама последовательность (разделитель - пробел). Элементы последовательности - целые числа, не превосходящие 10000 по модулю.

Выходные данные

Требуется вывести наибольшую возрастающую подпоследовательность данной последовательности. Если таких подпоследовательностей несколько, необходимо вывести одну (любую) из них.

|  |
| --- |
| from collections import deque  from typing import List, Iterable, Sequence  def longest\_increasing\_subsequence(sequence: Sequence[int]) -> Iterable[int]:  n = len(sequence)  distance\_table: List[int] = [1] \* n # Массив для хранения длины LIS до каждого элемента  prev\_index: List[int] = [-1] \* n # Массив для восстановления последовательности  # Заполняем массив dp  for i in range(n):  for j in range(i):  if sequence[j] < sequence[i] and distance\_table[j] + 1 > distance\_table[i]:  distance\_table[i] = distance\_table[j] + 1  prev\_index[i] = j  # Находим максимальную длину и индекс последнего элемента LCS  max\_length: int = max(distance\_table)  max\_index: int = distance\_table.index(max\_length)  # Восстанавливаем саму последовательность  result = deque()  while max\_index != -1:  result.append(sequence[max\_index])  max\_index = prev\_index[max\_index]  result.reverse()  return result  def main() -> None:  \_ = int(input())  sequence = list(map(int, input().split()))  lis = longest\_increasing\_subsequence(sequence)  # Выводим результат  print(' '.join(map(str, lis)))  if \_\_name\_\_ == "\_\_main\_\_":  main() |



Задание 5.

Дано N предметов массой m1, …, mN и стоимостью c1, …, cN соответственно.

Ими наполняют рюкзак, который выдерживает вес не более M. Определите набор предметов, который можно унести в рюкзаке, имеющий наибольшую стоимость.

Входные данные

В первой строке вводится натуральное число N, не превышающее 100 и натуральное число M, не превышающее 10000.

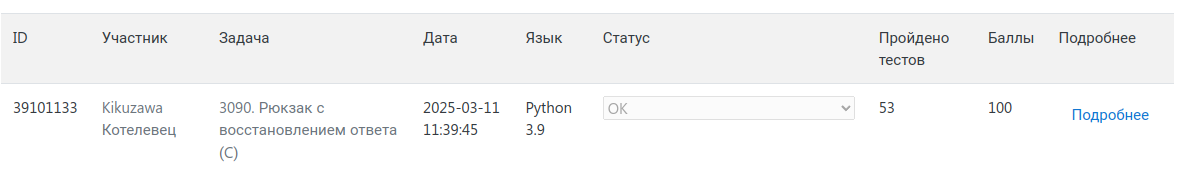
Во второй строке вводятся N натуральных чисел mi, не превышающих 100.

В третьей строке вводятся N натуральных чисел сi, не превышающих 100.

Выходные данные

Выведите номера предметов (числа от 1 до N), которые войдут в рюкзак наибольшей стоимости.

|  |
| --- |
| from itertools import product  from collections import deque  from typing import Iterable, Sequence, List  def knapsack(n: int, m: int, weights: Sequence[int], costs: Sequence[int]) -> Iterable[int]:  # Инициализация таблицы для динамического программирования  distance\_table: List[List[int]] = [[0] \* (m + 1) for \_ in range(n + 1)]  # Заполнение таблицы  for i, j in product(range(1, n + 1), range(m + 1)):  distance\_table[i][j] = distance\_table[i - 1][j] # Не берем текущий предмет  if j >= weights[i] and distance\_table[i - 1][j - weights[i]] + costs[i] > distance\_table[i][j]:  distance\_table[i][j] = distance\_table[i - 1][j - weights[i]] + costs[i] # Берем текущий предмет  # Восстановление набора предметов  selected\_items: deque[int] = deque()  remaining\_weight: int = m  for i in range(n, 0, -1):  if distance\_table[i][remaining\_weight] != distance\_table[i - 1][remaining\_weight]: # Если предмет n был выбран  selected\_items.append(i) # Добавляем номер предмета  remaining\_weight -= weights[i] # Уменьшаем оставшийся вес  selected\_items.reverse()  return selected\_items  def main() -> None:  n, m = map(int, input().split())  weights: List[int] = [0] + list(map(int, input().split()))  costs: List[int] = [0] + list(map(int, input().split()))  selected\_items = knapsack(n, m, weights, costs)  # Выводим результат  print(' '.join(map(str, selected\_items)))  if \_\_name\_\_ == "\_\_main\_\_":  main() |



Задание 6.

Дана строка, составленная из круглых, квадратных и фигурных скобок. Определите, какое наименьшее количество символов необходимо удалить из этой строки, чтобы оставшиеся символы образовывали правильную скобочную последовательность.

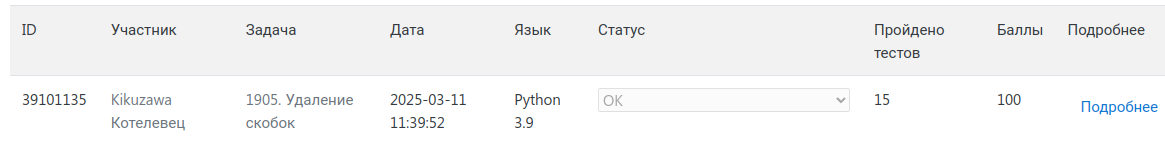
Входные данные

Строка из круглых, квадратных и фигурных скобок. Длина строки не превосходит 100 символов.

Выходные данные

Выведите строку максимальной длины, являющуюся правильной скобочной последовательностью, которую можно получить из исходной строки удалением некоторых символов.Если возможных ответов несколько, выведите любой из них.

|  |
| --- |
| from typing import Final, Dict, List, Sequence, AnyStr  BRACKET\_PAIRS: Final[Dict[str, str]] = {  '(': ')',  '[': ']',  '{': '}'  }  def reconstruct(  s: AnyStr,  distance\_table: Sequence[Sequence[int]],  table\_for\_restoring\_sequence: Sequence[Sequence[int]],  l: int,  r: int  ) -> AnyStr:  if distance\_table[l][r] == r - l + 1: # Все символы уже в правильной последовательности  return ""  if distance\_table[l][r] == 0: # Неправильная последовательность  return s[l:r + 1]  if table\_for\_restoring\_sequence[l][r] == -1: # Если нет разделения  return s[l] + reconstruct(s, distance\_table, table\_for\_restoring\_sequence, l + 1, r - 1) + s[r]  return (reconstruct(s, distance\_table, table\_for\_restoring\_sequence, l, table\_for\_restoring\_sequence[l][r]) +  reconstruct(s, distance\_table, table\_for\_restoring\_sequence, table\_for\_restoring\_sequence[l][r] + 1, r))  def fill\_tables(  s: AnyStr,  n: int,  distance\_table: List[List[int]],  table\_for\_restoring\_sequence: List[List[int]]  ) -> None:  for right in range(n):  for left in range(right, -1, -1):  if left == right:  distance\_table[left][right] = 1  else:  min\_removals = float('inf')  split\_index = -1  # Проверка на соответствие скобок  if s[left] in BRACKET\_PAIRS and s[right] == BRACKET\_PAIRS[s[left]]:  min\_removals = distance\_table[left + 1][right - 1]  # Разделение на подзадачи  for k in range(left, right):  current\_removals = distance\_table[left][k] + distance\_table[k + 1][right]  if min\_removals > current\_removals:  min\_removals = current\_removals  split\_index = k  distance\_table[left][right] = min\_removals  table\_for\_restoring\_sequence[left][right] = split\_index  def min\_removals\_to\_valid\_parentheses(s: AnyStr) -> AnyStr:  n: int = len(s)  # Таблица для хранения минимального количества удалений  # Один символ всегда является правильной последовательностью  distance\_table: List[List[int]] = [[1 if i == j else 0 for j in range(n)] for i in range(n)]  # Таблица для восстановления последовательности  table\_for\_restoring\_sequence: List[List[int]] = [[0] \* n for \_ in range(n)]  fill\_tables(s, n, distance\_table, table\_for\_restoring\_sequence)  return reconstruct(s, distance\_table, table\_for\_restoring\_sequence, 0, n - 1)  def main() -> None:  s: str = input()  result = min\_removals\_to\_valid\_parentheses(s)  print(result)  if \_\_name\_\_ == "\_\_main\_\_":  main() |



Задание 7.

Палиндромом называется строка, которая одинаково читается как слева направо, так и справа налево. Подпалиндромом данной строки называется последовательность символов из данной строки, не обязательно идущих подряд, являющаяся палиндромом. Например, HELOLEH является подпалиндромом строки HTEOLFEOLEH. Напишите программу, находящую в данной строке подпалиндром максимальной длины.

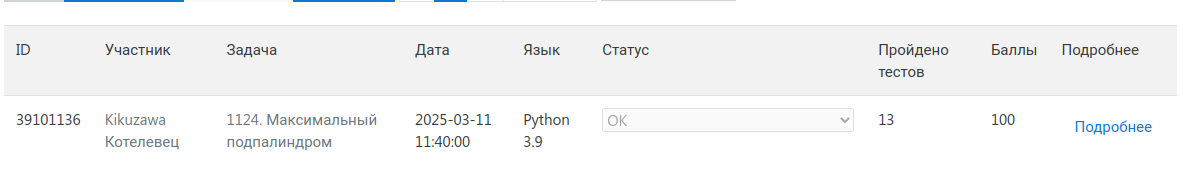
Входные данные

Во входном файле находится строка длиной не более 100 символов, состоящая из заглавных букв латинского алфавита.

Выходные данные

Выведите на первой строке выходного файла длину максимального подпалиндрома, а на второй строке сам максимальный подпалиндром. Если таких подпалиндромов несколько, то ваша программа должна вывести любой из них.

|  |
| --- |
| from typing import Tuple, List  from collections import deque  def fill\_table(n: int, s: str, distance\_table: List[List[int]]) -> None:  for length in range(2, n + 1): # Длина подстроки  for i in range(n - length + 1):  j = i + length - 1  if s[i] == s[j]:  distance\_table[i][j] = distance\_table[i + 1][j - 1] + 2  else:  distance\_table[i][j] = max(distance\_table[i + 1][j], distance\_table[i][j - 1])  def longest\_palindromic\_subsequence(s: str) -> Tuple[int, str]:  n = len(s)  # Таблица для хранения длины максимального подпалиндрома  # Все одиночные символы являются палиндромами длины 1  distance\_table: List[List[int]] = [[1 if i == j else 0 for j in range(n)] for i in range(n)]  fill\_table(n, s, distance\_table)  # Длина максимального подпалиндрома  max\_length: int = distance\_table[0][n - 1]  # Восстановление самого подпалиндрома  left\_index: int = 0  right\_index: int = n - 1  subsequence: deque[str] = deque()  while left\_index <= right\_index:  if s[left\_index] == s[right\_index]:  subsequence.append(s[left\_index])  left\_index += 1  right\_index -= 1  elif distance\_table[left\_index + 1][right\_index] >= distance\_table[left\_index][right\_index - 1]:  left\_index += 1  else:  right\_index -= 1  # Если длина подпалиндрома четная, то мы добавляем его в обратном порядке  # Если нечетная, то добавляем последний символ  palindromic\_subsequence = ''.join(subsequence)  if len(palindromic\_subsequence) \* 2 == max\_length:  result = palindromic\_subsequence + palindromic\_subsequence[::-1]  else:  result = palindromic\_subsequence + palindromic\_subsequence[-2::-1]  return max\_length, result  def main() -> None:  s = input()  max\_length, palindromic\_subsequence = longest\_palindromic\_subsequence(s)  print(max\_length)  print(palindromic\_subsequence)  if \_\_name\_\_ == "\_\_main\_\_":  main() |



Задание 8.

Как-то раз гному Кварку попала в руки карта сокровищ. На карте отмечено NN точек, в которых может находиться клад. Все точки пронумерованы числами от 11 до NN. Для каждой пары точек Кварк знает длину дороги, их соединяющей. Свои поиски Кварк начинает от точки с номером 11. Прежде чем начать свой долгий путь, хитрый гном вычеркивает точки, в которых, по его мнению, клада быть не может. Гарантируется, что точка с номером 11 никогда не бывает вычеркнута. После этого Кварк выбирает некоторый маршрут, проходящий через все оставшиеся на карте точки. Маршрут не проходит через одну и ту же точку более одного раза. Кварк может ходить только по дорогам, соединяющим невычеркнутые точки.

Кварк хочет выбрать маршрут минимальной длины. Необходимо найти такой маршрут для Кварка.

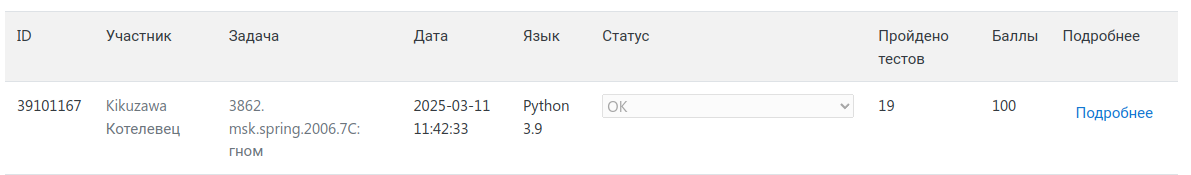
Входные данные

В первой строке находится одно целое число NN (1<N≤15)(1<N≤15) — количество точек, отмеченных на карте. В последующих NN строках находятся расстояния между точками. В (i+1)(i+1)-й строке находятся NN целых чисел di1,di2,…,diNdi1,di2,…,diN — длины дорог от ii-й точки до всех остальных. Гарантируется, что dij=djidij=dji, dii=0dii=0 и 0<dij<1000<dij<100. В (N+2)(N+2)-й строке находится одно целое число QQ (1<Q≤1000)(1<Q≤1000) — количество вариантов вычеркивания точек для данной карты. В последующих QQ строках содержится описание вариантов вычеркивания. Описание начинается с числа CC (0≤C<N)(0≤C<N) — количества точек, в которых, по мнению Кварка, клада быть не может. Следующие CC чисел задают номера этих точек.

Выходные данные

Выведите QQ строк. В каждой строке выведите одно целое число — длину минимального маршрута при соответствующем варианте вычеркивания точек.

|  |
| --- |
| def min\_path(mask, pos, n, dist):  # Базовый случай: если все точки посещены, возвращаемся в точку 11  if mask == ((1 << n) - 1):  return dist[pos][11]  ans = float('inf')  # Проверяем все непосещенные точки  for i in range(n):  if (mask & (1 << i)) == 0: # Если точка не посещена  # Вычисляем расстояние через текущую точку  ans = min(ans, dist[pos][i] + min\_path(mask | (1 << i), i, n, dist))  return ans  # Чтение входных данных  N = int(input())  dist = []  for \_ in range(N):  dist.append(list(map(int, input().split())))  Q = int(input())  results = []  # Обработка каждого варианта удаления точек  for \_ in range(Q):  # Читаем количество удаляемых точек и сами точки  C = list(map(int, input().split()))  count = C[0] # Количество точек для удаления  points = C[1:] # Список точек для удаления  # Создаём маску активных точек  active\_mask = (1 << N) - 1  for point in points:  active\_mask &= ~(1 << (point - 1)) # Приводим к нулевому индексу  # Если точка 11 не удалена и есть хотя бы одна активная точка  if active\_mask & (1 << 11):  result = min\_path(active\_mask, 11, N, dist)  results.append(str(result))  print('\n'.join(results)) |



Задание 9

Даны N золотых слитков известных масс. Определите, какую наибольшую массу золота можно унести, если вместимость рюкзака не превышает S.

Входные данные

Программа получает на вход целое число S — вместимость рюкзака, не превосходящее 10000 и количество слитков N, не превосходящее 300. Далее следует N целых неотрицательных чисел, не превосходящих 100000 — веса слитков.

Выходные данные

Программа должна вывести единственное целое число — максимально возможных вес золота, который поместится в данный рюкзак.

|  |
| --- |
| from typing import List  def max\_gold\_weight(capacity: int, weights: list) -> int:  achievable\_weights: List[int] = [0] \* (capacity + 1)  achievable\_weights[0] = 1 # Нулевой вес всегда достижим  # Проходим по каждому слитку  for weight in weights:  # Обновляем массив достижимых весов в обратном порядке  for current\_capacity in range(capacity, weight - 1, -1):  if achievable\_weights[current\_capacity - weight] == 1:  achievable\_weights[current\_capacity] = 1  # Находим максимальный вес, который можно унести  for i in range(capacity, -1, -1):  if achievable\_weights[i] == 1:  return i # Возвращаем максимальный вес  return 0 # Если ничего не удалось унести  def main() -> None:  capacity, n = map(int, input().split())  weights: List[int] = list(map(int, input().split()))  max\_weight: int = max\_gold\_weight(capacity, weights)  print(max\_weight)  if \_\_name\_\_ == "\_\_main\_\_":  main() |

Лабораторная работа 4

Задание 1

На стандартной шахматной доске (8×88×8) живут 2 шахматных коня: Красный и Зелёный. Обычно они беззаботно скачут по просторам доски, пощипывая шахматную травку, но сегодня особенный день: у Зелёного коня день рождения. Зелёный конь решил отпраздновать это событие вместе с Красным. Но для осуществления этого прекрасного плана им нужно оказаться на одной клетке. Заметим, что Красный и Зёленый шахматные кони сильно отличаются от черного с белым: они ходят не по очереди, а одновременно, и, если оказываются на одной клетке, никто никого не съедает. Сколько ходов им потребуется, чтобы насладиться праздником?

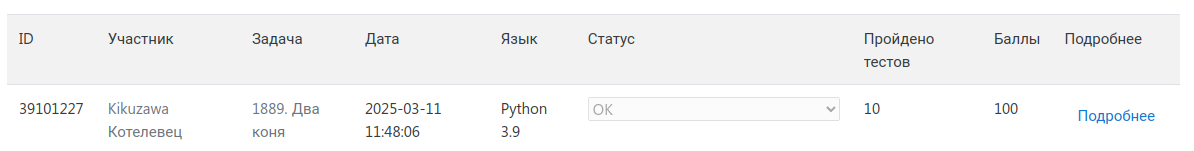
**Входные данные**

Во входном файле содержатся координаты коней, записанные по стандартным шахматным правилам (т. е. двумя символами — маленькая латинская буква (от a до h) и цифра (от 1 до 8), задающие столбец и строку соответственно).

**Выходные данные**

Выходной файл должен содержать наименьшее необходимое количество ходов, либо −1−1, если кони не могут встретиться.

|  |
| --- |
| from collections import deque  from itertools import product  from typing import Tuple, Generator, Any, Set  def get\_moves(x: int, y: int) -> Generator[Tuple[int, int], Any, None]:  moves: Tuple[Tuple[int, int], ...] = (  (x + 1, y + 2), (x + 2, y + 1), (x + 2, y - 1), (x + 1, y - 2),  (x - 1, y - 2), (x - 2, y - 1), (x - 2, y + 1), (x - 1, y + 2)  )  yield from ((mx, my) for mx, my in moves if 0 <= mx <= 7 and 0 <= my <= 7)  def knights\_meet(knight1: Tuple[int, int], knight2: Tuple[int, int]) -> int:  visited: Set[Tuple[Tuple[int, int], Tuple[int, int]]] = set()  queue: deque[Tuple[Tuple[int, int], Tuple[int, int], int]] = deque([(knight1, knight2, 0)])  while queue:  k1, k2, moves = queue.popleft()  if k1 == k2:  return moves  visited.add((k1, k2))  for move1, move2 in product(get\_moves(k1[0], k1[1]), get\_moves(k2[0], k2[1])):  if (move1, move2) not in visited:  visited.add((move1, move2)) # Добавляем в visited сразу  queue.append((move1, move2, moves + 1))  return -1  def main() -> None:  knight1, knight2 = map(lambda k: (ord(k[0]) - ord('a'), int(k[1]) - 1), input().split())  result: int = knights\_meet(knight1, knight2)  print(result)  if \_\_name\_\_ == "\_\_main\_\_":  main() |



Задание 2

На банкет были приглашены N Очень Важных Персон (ОВП). Были поставлены 2 стола. Столы достаточно большие, чтобы все посетители банкета могли сесть за любой из них. Проблема заключается в том, что некоторые ОВП не ладят друг с другом и не могут сидеть за одним столом. Вас попросили определить, возможно ли всех ОВП рассадить за двумя столами.

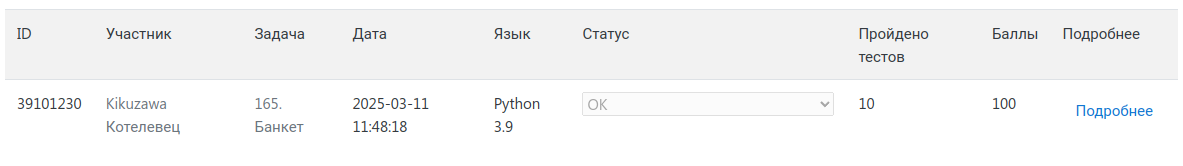
Входные данные

В первой строке входных данных содержатся два числа: N и M (1 <= N,M <= 100), где N – количество ОВП, а M – количество пар ОВП, которые не могут сидеть за одним столом. В следующих M строках записано по 2 числа – пары ОВП, которые не могут сидеть за одним столом.

Выходные данные

Если способ рассадить ОВП существует, то выведите YES в первой строке и номера ОВП, которых необходимо посадить за первый стол, во второй строке. В противном случае в первой и единственной строке выведите NO.

|  |
| --- |
| from collections import deque  from typing import List, cast, Tuple  class Graph:  def \_\_init\_\_(self, n: int) -> None:  """Инициализируем граф с n вершинами."""  self.\_n = n  self.\_adj\_list: List[List[int]] = [[] for \_ in range(n)] # Список смежности для графа  self.\_color: List[int] = []  def add\_edge(self, u: int, v: int) -> None:  """Добавляем ребро между вершинами u и v."""  self.\_adj\_list[u - 1].append(v - 1)  self.\_adj\_list[v - 1].append(u - 1)  def is\_bipartite(self) -> bool:  self.\_color = [-1] \* self.\_n # -1: не окрашена, 0: первая доля, 1: вторая доля  def bfs(start\_inner: int) -> bool:  queue: deque[int] = deque([start\_inner])  self.\_color[start\_inner] = 0  while queue:  v = queue.popleft()  for u in self.\_adj\_list[v]:  if self.\_color[u] == -1:  self.\_color[u] = 1 - self.\_color[v]  queue.append(u)  elif self.\_color[u] == self.\_color[v]:  return False  return True  for start in range(self.\_n):  if self.\_color[start] == -1 and not bfs(start): # Если вершина не посещена  return False  return True  @property  def bipartite\_partition(self) -> List[int]:  if self.is\_bipartite():  return [i + 1 for i in range(self.\_n) if self.\_color[i] == 0]  return []  def main() -> None:  n, m = map(int, input().split())  pairs = cast(List[Tuple[int, int]], [tuple(map(int, input().split())) for \_ in range(m)])  # Создаем граф и добавляем ребра  graph = Graph(n)  for u, v in pairs:  graph.add\_edge(u, v)  # Проверяем, возможно ли рассадить  result = graph.bipartite\_partition  if result:  print("YES")  print(\*result)  else:  print("NO")  if \_\_name\_\_ == "\_\_main\_\_":  main() |



Задание 3

один из нескольких возможных рельефов, для простоты ограничимся тремя видами рельефов - поле, лес и вода. Поселенец перемещается по карте, при этом на перемещение в клетку, занятую полем, необходима одна единица времени, на перемещение в лес - две единицы времени, а перемещаться в клетку с водой нельзя.

У вас есть один поселенец, вы определили место, где нужно построить город, чтобы как можно скорее завладеть всем миром. Найдите маршрут переселенца, приводящий его в место строительства города, требующий минимального времени. На каждом ходе переселенец может перемещаться в клетку, имеющую общую сторону с той клеткой, где он сейчас находится.

Входные данные

Во входном файле записаны два натуральных числа N и M, не превосходящих 1000 - размеры карты мира (N - число строк в карте, M - число столбцов). Затем заданы координаты начального положения поселенца x и y, где x - номер строки, y - номер стролбца на карте (1 ≤ x ≤ N, 1 ≤ y ≤ M), строки нумеруются сверху вниз, столбцы - слева направо. Затем аналогично задаются координаты клетки, куда необходимо привести поселенца.

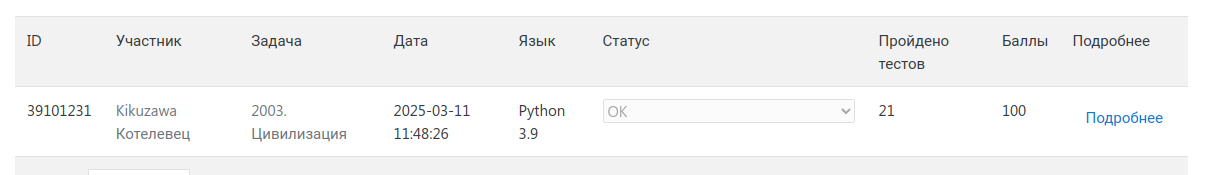
Далее идет описание карты мира в виде N строк, каждая из которых содержит M символов. Каждый символ может быть либо “.” (точка), обозначающим поле, либо “W”, обозначающим лес, либо “#”, обозначающим воду. Гарантируется, что начальная и конечная клетки пути переселенца не являются водой.

Выходные данные

В первой строке выходного файла выведите количество единиц времени, необходимое для перемещения поселенца (перемещение в клетку с полем занимает 1 единицу времени, перемещение в клетку с лесом - 2 единицы времени). Во второй строке выходного файла выведите последовательность символов, задающих маршрут переселенца. Каждый символ должен быть одним из четырех следующих: “N” (движение вверх), “E” (движение вправо), “S” (движение вниз), “W” (движение влево). Если таких маршрутов несколько, выведите любой из них.

Если дойти из начальной клетки в конечную невозможно, выведите число -1.

|  |
| --- |
| from collections import deque  from typing import List, cast, Tuple  class Graph:  def \_\_init\_\_(self, n: int) -> None:  """Инициализируем граф с n вершинами."""  self.\_n = n  self.\_adj\_list: List[List[int]] = [[] for \_ in range(n)] # Список смежности для графа  self.\_color: List[int] = []  def add\_edge(self, u: int, v: int) -> None:  """Добавляем ребро между вершинами u и v."""  self.\_adj\_list[u - 1].append(v - 1)  self.\_adj\_list[v - 1].append(u - 1)  def is\_bipartite(self) -> bool:  self.\_color = [-1] \* self.\_n # -1: не окрашена, 0: первая доля, 1: вторая доля  def bfs(start\_inner: int) -> bool:  queue: deque[int] = deque([start\_inner])  self.\_color[start\_inner] = 0  while queue:  v = queue.popleft()  for u in self.\_adj\_list[v]:  if self.\_color[u] == -1:  self.\_color[u] = 1 - self.\_color[v]  queue.append(u)  elif self.\_color[u] == self.\_color[v]:  return False  return True  for start in range(self.\_n):  if self.\_color[start] == -1 and not bfs(start): # Если вершина не посещена  return False  return True  @property  def bipartite\_partition(self) -> List[int]:  if self.is\_bipartite():  return [i + 1 for i in range(self.\_n) if self.\_color[i] == 0]  return []  def main() -> None:  n, m = map(int, input().split())  pairs = cast(List[Tuple[int, int]], [tuple(map(int, input().split())) for \_ in range(m)])  # Создаем граф и добавляем ребра  graph = Graph(n)  for u, v in pairs:  graph.add\_edge(u, v)  # Проверяем, возможно ли рассадить  result = graph.bipartite\_partition  if result:  print("YES")  print(\*result)  else:  print("NO")  if \_\_name\_\_ == "\_\_main\_\_":  main() |



Задание 4.

Предприятие «Авто-2010» выпускает двигатели для известных во всём мире автомобилей. Двигатель состоит ровно из n

деталей, пронумерованных от 1 до n

, при этом деталь с номером i

изготавливается за pi

секунд. Специфика предприятия «Авто-2010» заключается в том, что там одновременно может изготавливаться лишь одна деталь двигателя. Для производства некоторых деталей необходимо иметь предварительно изготовленный набор других деталей.

Генеральный директор «Авто-2010» поставил перед предприятием амбициозную задачу — за наименьшее время изготовить деталь с номером 1, чтобы представить её на выставке.

Требуется написать программу, которая по заданным зависимостям порядка производства между деталями найдёт наименьшее время, за которое можно произвести деталь с номером 1.

Входные данные

Первая строка входного файла содержит число n

(1≤n≤100000

) — количество деталей двигателя. Вторая строка содержит n

натуральных чисел p1,p2,…,pn

, определяющих время изготовления каждой детали в секундах. Время для изготовления каждой детали не превосходит 109

секунд.

Каждая из последующих n

строк входного файла описывает характеристики производства деталей. Здесь i

-я строка содержит число деталей ki

, которые требуются для производства детали с номером i

, а также их номера. В i

-й строке нет повторяющихся номеров деталей. Сумма всех чисел ki

не превосходит 200000.

Известно, что не существует циклических зависимостей в производстве деталей.

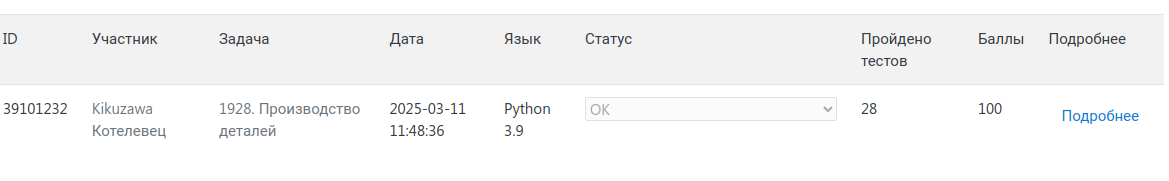
Выходные данные

В первой строке выходного файла должны содержаться два числа: минимальное время (в секундах), необходимое для скорейшего производства детали с номером 1 и число k

деталей, которые необходимо для этого произвести. Во второй строке требуется вывести через пробел k

чисел — номера деталей в том порядке, в котором следует их производить для скорейшего производства детали с номером 1.

|  |
| --- |
| from typing import List, Tuple, Sequence  from collections import deque  def produce\_part(  part\_id: int,  production\_times: Sequence[int],  dependencies: Sequence[Sequence[int]],  produced: List[bool],  production\_order: deque[int]  ) -> int:  if produced[part\_id]:  return 0  produced[part\_id] = True  total\_time = production\_times[part\_id] # Время на изготовление текущей детали  for dependency in dependencies[part\_id - 1]: # Индексация зависит от 0  total\_time += produce\_part(dependency, production\_times, dependencies, produced, production\_order)  production\_order.append(part\_id)  return total\_time  def calculate\_minimum\_production\_time(  n: int,  production\_times: Sequence[int],  dependencies: Sequence[Sequence[int]]  ) -> Tuple[int, Sequence[int]]:  produced: List[bool] = [False] \* (n + 1)  production\_order: deque[int] = deque()  total\_time: int = produce\_part(1, production\_times, dependencies, produced, production\_order)  return total\_time, production\_order  def main() -> None:  n: int = int(input())  production\_times: List[int] = [0] + list(map(int, input().split()))  dependencies: List[List[int]] = [list(map(int, input().split()[1:])) for \_ in range(n)]  total\_time, order = calculate\_minimum\_production\_time(n, production\_times, dependencies)  print(total\_time, len(order))  print(\*order)  if \_\_name\_\_ == "\_\_main\_\_":  main() |



Задание 5.