Заключительный этап научно–образовательного соревнования олимпиады школьников «Шаг в будущее» по профилю «Инженерное дело» специализации «Профессор Лебедев» (общеобразовательный предмет «информатика»)

Типовой вариант задания для 11 класса

Задание 1

Расставьте операции сложения и умножения в строке так, чтобы получилось верное равенство

На вход подаётся строка, содержащая числа, записанные через произвольное число пробелов. Первым числом указывается результат искомого выражения. Результатом ожидается строка, содержащая знаки операций

Входные данные: 24 1 2 3 4

Результат: = * * *

Комментарий: 24 = 1*2*3*4

Входные данные: 5 3 2

Результат: = +

Комментарий: 5 = 3*2

Задание 2

Из входной строки, содержащей произвольное количество слов (последовательности символов, записанных через пробел), необходимо удалить, сохраняя пробелы, все слова, чья длина равна значению факториала некоего целого числа.

Входные данные: «123 йцукенг й йцу йцукенгшщз зщшг ойойой»

Результат: «123 йцукенг йцу йцукенгшщз зщшг »

Комментарий: длина «й» = 1!, длина «ойойой» = 3!

Задание 3

Из входной строки, содержащей произвольное количество слов (последовательности символов, записанных через пробел), необходимо создать строку, удвоив количество пробелов между соседними словами, для которых расстояние Левенштейна меньше 3.
Примечание:

Левенштейна – это минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую. Например, расстояние Левенштейна между словом «мама» и «папа» будет 2 (2 замены), а между «ель» и «гнёт» – 4 (3 замены и 1 вставка).

Задание 4

На вход подаётся строка, содержащая целые числа. Необходимо найти матрицу, чей определитель будет наибольшим. Результатом программы должна быть строка, содержащая искомую матрицу, записанную построчно.

Входная строка: 1 2 5 6 7

Результат: 7 1 2 6

Задание 5

На вход подаётся строка, содержащая записанные через пробел целые числа. Необходимо найти наибольшую сумму чисел, расположенных между числами, имеющими индексы, равные соседним числам Фибоначчи (граничные числа в сумме учитываются).

Примечание: числа Фибоначчи – последовательность чисел, где следующее число равно сумме двух предыдущих: 0,1,1,2,3,5,8 …

Входная строка: 8 2 4 3 9 1 2 3 4 5

Результат: 13

Комментарий к ответу: [3;5]3е число это 3, 5е число это 1, 3+9+1 = 13; [5;8] = 1+2+3+4=10

Задание 6

На вход подаётся строка, содержащая записанные через пробел целые числа. Необходимо найти наибольшее количество чисел, сумма цифр которых является одинаковым простым числом.

Входная строка: 19 23 113 302 104 3 12

Результат: 4

Решение типового варианта для 11 класса

Распределение баллов по заданиям:

Номер задачи	1	2	3	4	5	6
Баллы	5	15	15	18	22	25

Задание 1

Расставьте операции сложения и умножения в строке так, чтобы получилось верное равенство

На вход подаётся строка, содержащая числа, записанные через произвольное число пробелов. Первым числом указывается результат искомого выражения. Результатом ожидается строка, содержащая знаки операций

Входные данные: 24 1 2 3 4

Результат: = * * *

Комментарий: 24 = 1*2*3*4

Входные данные: 5 3 2

Результат: = +

Комментарий: 5 = 3*2

```
def operations(num, length):
    res = []

    for i in range(length):
        if num % 2 == 0:
            res.append(0)
        else:
            res.append(1)
        num = num // 2

    return res[::-1]

# найдем все числа, которые должны получиться в процессе умножения
# и перепишем те числа, которые просто складываются
def count_multiplication(nums_array, ops_array):
    curr_nums = [nums_array[0]]
    curr_num_count = 0
    prev_sum = False

    for i in range(len(ops_array)):
        # если умножение
        if (ops_array[i] == 1):
            # начало нового этапа умножения, число нужно добавить
            if (prev_sum):
                curr_nums.append(nums_array[i] * nums_array[i+1])
```

```
            prev_sum = False
            curr_num_count += 1
        # последовательное умножение
        else:
            curr_nums[curr_num_count] *= nums_array[i+1]

    # если сложение
    else:
        prev_sum = True
        # последнее число складывается
        if i == len(ops_array) - 1:
            curr_nums.append(nums_array[-1])
        # будет сумма произведений, чисел добавлять не надо, но надо их разделить
        elif (ops_array[i+1] == 1):
            pass
        # число "не используется" в умножении, добавим его
        else:
            curr_nums.append(nums_array[i])
            curr_num_count += 1

  return curr_nums

def test_solution(nums_array, ops_array, awaiting_res):
  res = 0
  mult_result = count_multiplication(nums_array, ops_array)

  # оставшиеся числа просто сложим
  for elem in mult_result:
      res += elem
      if res > awaiting_res:
          return False

  if res == awaiting_res:
      return True
  return False


if __name__ == '__main__':
  numbers_arr = input().split()
  for i in range(0, len(numbers_arr)):
      numbers_arr[i] = int(numbers_arr[i])

  resulting_num = numbers_arr[0]
  checked_nums = numbers_arr[1:]

  # всего возможных вариантов расстановки знаков
  possible_variants = 2 ** (len(checked_nums) - 1)

  # каждый из этих вариантов представим как последовательность 0 и 1,
  # где 0 - сложение, 1 - умножение
  # это можно легко сделать, используя перевод чисел в двоичную систему
  variants = []
```

```
    for i in range(possible_variants):
        variants.append(operations(i, len(checked_nums) - 1))

    solution = []
    for variant in variants:
        if (test_solution(checked_nums, variant, resulting_num)):
            solution = variant
            break

    result_string = "="
    for num in solution:
        if num == 1:
            result_string += " *"
        else:
            result_string += " +"

    print(result_string)
```

Задание 2

Из входной строки, содержащей произвольное количество слов (последовательности символов, записанных через пробел), необходимо удалить, сохраняя пробелы, все слова, чья длина равна значению факториала некоего целого числа.

Входные данные: «123 йцукенг й йцу йцукенгшщз зщшг ойойой»

Результат: «123 йцукенг йцу йцукенгшщз зщшг »

Комментарий: длина «й» = 1!, длина «ойойой» = 3!

```
def test_if_factorial(num):
    fact = 1
    curr_step = 2

    while fact < num:
        fact *= curr_step
        curr_step += 1

    if fact == num:
        return True
    return False


if __name__ == '__main__':
    splitted_words = input().split()

    result_string = ""

    for i in range(len(splitted_words)):
        word = splitted_words[i]
        if not(test_if_factorial(len(word))):
            result_string += word
```

```
        if i != len(splitted_words) - 1:
            result_string += " "

    print(result_string)
    #print("--", result_string, "--", sep="")
```

Задание 3

Из входной строки, содержащей произвольное количество слов (последовательности символов, записанных через пробел), необходимо создать строку, удвоив количество пробелов между соседними словами, для которых расстояние Левенштейна меньше 3. Примечание:

Левенштейна – это минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую. Например, расстояние Левенштейна между словом «мама» и «папа» будет 2 (2 замены), а между «ель» и «гнёт» – 4 (3 замены и 1 вставка).

```
def lev_distance(str1, str2):
    len1 = len(str1)
    len2 = len(str2)

    if len1 == 1 and len2 == 1:
        if str1 == str2:
            return 0
        else:
            return 1
    elif (len1 == 0) and (len2 > 0):
        return len2
    elif (len2 == 0) and (len1 > 0):
        return len1

    last_sym = 0
    if str1[-1] != str2[-1]:
        last_sym = 1

    return min(lev_distance(str1, str2[:len2-1]) + 1,
               lev_distance(str1[:len1-1], str2) + 1,
               lev_distance(str1[:len1-1], str2[:len2-1]) + last_sym)


if __name__ == '__main__':
    splitted_words = input().split()

    result_string = ""

    for i in range(len(splitted_words) - 1):
        dist = lev_distance(splitted_words[i], splitted_words[i+1])
```

```
        result_string += splitted_words[i]
        if dist < 3:
            result_string +=  "  "
        else:
            result_string += " "

    result_string += splitted_words[-1]

    print(result_string)
```

Задание 4

На вход подаётся строка, содержащая целые числа. Необходимо найти матрицу, чей определитель будет наибольшим. Результатом программы должна быть строка, содержащая искомую матрицу, записанную построчно.

Входная строка: 1 2 5 6 7

Результат: 7 1 2 6

```
from math import sqrt

def get_all_permutatons(nums_array):
    if len(nums_array) == 1:
        return[nums_array]

    res = []
    for i in range(len(nums_array)):
        curr_num = nums_array[i]
        remaining = nums_array[:i] + nums_array[i+1:]

        for elems in get_all_permutatons(remaining):
            res.append(([curr_num] + elems))

    return res

def determinant_recursive(matrix, curr_res):
    size = len(matrix)

    if size == 1:
        return matrix[0][0]

    if size == 2:
       return curr_res * (matrix[0][0] * matrix[1][1] -
                          matrix[1][0] * matrix[0][1])

    # будем брать верхнюю строку матрицы и умножать на детерминант мартриц,
    # полученых из остальных строк
    # смотри алгоритм нахождения определителя матрицы N x N на Википедии
    else:
        sign = -1
```

```
        sum = 0

        for i in range(size):
            add_minor = []
            sign *= -1
            curr_mult = sign * matrix[0][i]

            for j in range(1, size):
                buff = []
                for k in range(size):
                    # столбец, в котором выбранный элемент, должен быть убран
                    if i != k:
                        buff.append(matrix[j][k])
                add_minor.append(buff)

            sum += curr_res * \
                determinant_recursive(add_minor, curr_mult)

        return sum

def to_square_matrix(arr, size):
    res = [0] * size
    for i in range(size):
        res[i] = [0] * size

    for i in range(size):
        for j in range(size):
            res[i][j] = arr[i * size + j]

    return res

if __name__ == '__main__':
    numbers_arr = input().split()
    for i in range(0, len(numbers_arr)):
        numbers_arr[i] = int(numbers_arr[i])

    nums_am = len(numbers_arr)

    max_matr_size = int(sqrt(nums_am))

    # найдем все возможные перестановки полученных чисел
    all_permutation = get_all_permutatons(numbers_arr)

    best_matrix = numbers_arr[0]
    best_determinant = numbers_arr[0]

    # от всех возможных перестановок будем "отрезать" нужное нам для создания квадратной
матрицы
    # число цифр, и будем находить детерминант матрицы, созданной из этих цифр
    for curr_size in range(max_matr_size + 1):
        for elem in all_permutation:
            matr = to_square_matrix(elem, curr_size)
```

```
        determ = determinant_recursive(matr, 1)

        if determ >= best_determinant:
            best_determinant = determ
            best_matrix = elem[:curr_size*curr_size]

    # print(best_determinant)
    for elem in best_matrix:
        print(elem, end=" ")
```

Задание 5

На вход подаётся строка, содержащая записанные через пробел целые числа. Необходимо найти наибольшую сумму чисел, расположенных между числами, имеющими индексы, равные соседним числам Фибоначчи (граничные числа в сумме учитываются).

Примечание: числа Фибоначчи – последовательность чисел, где следующее число равно сумме двух предыдущих: 0,1,1,2,3,5,8 …

Входная строка: 8 2 4 3 9 1 2 3 4 5

Результат: 13

Комментарий к ответу: [3;5]3е число это 3, 5е число это 1, 3+9+1 = 13; [5;8] = 1+2+3+4=10

```
# создадим массив из чисел Фибоначчи, которые не превышают размер массива
# чтобы использовать для индексации
def create_Fibonacci_arr(nums_amount):
    arr = [0, 1]
    counter = 1
    i = 1
    while i <= nums_amount - 1:
        arr.append(i)
        counter += 1
        i = arr[counter] + arr[counter - 1]

    return arr

if __name__ == '__main__':

    numbers_arr = input().split()
    for i in range(0, len(numbers_arr)):
        numbers_arr[i] = int(numbers_arr[i])

    nums_amount = len(numbers_arr)

    fibonacci_arr = create_Fibonacci_arr(nums_amount)
    # print(fibonacci_arr)

    # первые числа Фибоначчи - 0 и 1, поэтому начальное значение
    # максимальной суммы элементов - сумма 0-го и 1-го элемента введенного массива
    max_sum = numbers_arr[fibonacci_arr[0]] + numbers_arr[fibonacci_arr[1]]
```

```
    # последовательно берем пары элементов из массива чисел Фибоначчи
    # и суммируем элементы введенного массива, используя эти пары как индексы
    # полученную сумму сравнивае с текущей максимальной
    for i in range(1, len(fibonacci_arr) - 1):
        curr_sum = 0
        for j in range(fibonacci_arr[i], fibonacci_arr[i+1] + 1):
            curr_sum += numbers_arr[j]

        if curr_sum > max_sum:
            max_sum = curr_sum

    print(max_sum)
```

Задание 6

На вход подаётся строка, содержащая записанные через пробел целые числа. Необходимо найти наибольшее количество чисел, сумма цифр которых является одинаковым простым числом.

Входная строка: 19 23 113 302 104 3 12

Результат: 4

```
from math import  sqrt

# найдем сумму цифр числа
# будем добавлять цифры "с конца", используя остаток от деления на 10
# после чего "отрежем" эту цифру с помощью целочисленного деления
# (другой вариант решения подзадачи - представить число как строку и
# переводить каждый char в цифру отдельно)
def sum_of_digits(num):
    sum = 0

    while num > 0:
        sum += num % 10
        num = num // 10
    return sum

# используем "топорный" алгоритм, так как максимальная оптимизация не требуется
# будем проверять делимость числа на все целые числа, меньше корня этого числа,
# хотя достаточно проверять делимость только на меньшие простые
def test_if_prime(num):
    for i in range(2, round(sqrt(num)) + 1):
        a = num / i
        b = num // i
        if a == b:
            return False
    return True

if __name__ == '__main__':
    numbers_arr = input().split()
    for i in range(0, len(numbers_arr)):
```

```
    numbers_arr[i] = int(numbers_arr[i])

# создадим массив, где будем записывать встреченные нами простые суммы цифр
prime_sums = []
for curr_number in numbers_arr:
    curr_digit_sum = sum_of_digits(curr_number)
    if (test_if_prime(curr_digit_sum)):
        prime_sums.append(curr_digit_sum)
# print(prime_sums)

max_prime_count = 0
# посчитаем кол-во вхождений каждого из простых чисел, чтобы найти наибольшее
# для простоты даже не будем проверять, считали ли мы уже сумму вхождений
# для этого числа
for curr_number in prime_sums:
    count = 0
    for checked_number in prime_sums:
        if curr_number == checked_number:
            count += 1

    if count > max_prime_count:
        max_prime_count = count

print(max_prime_count)
```