САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1

по курсу «Алгоритмы и структуры данных»

Тема: Greedy and DP  
Вариант 18

Выполнил:

Лазарев Марк Олегович

К3241

Санкт-Петербург

2025 г.

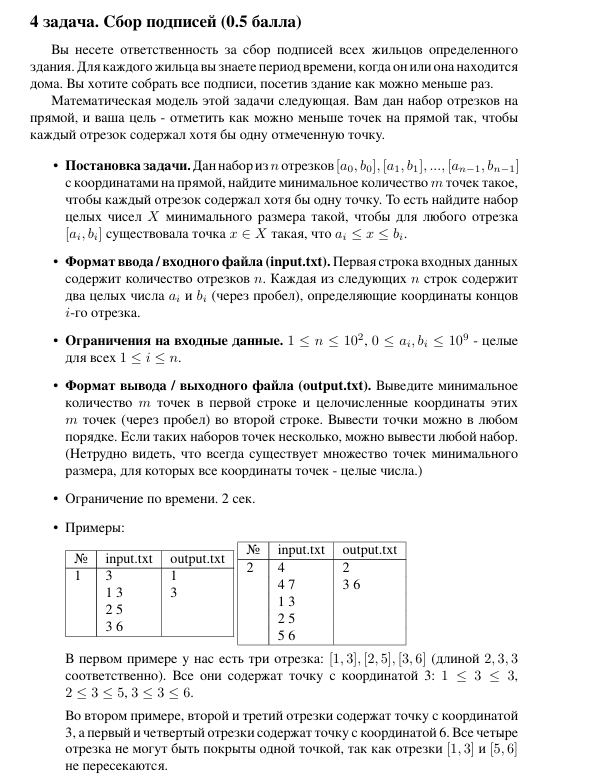
[**Задачи по варианту**](#_hgvy6a6e44x7)

[Задача №1.](#_p1ur54b6c1ty) Сбор подписей  
[Задача №2.](#_y9wyx2jesy3p) Максимальное количество призов

[Задача №3.](#_p1ur54b6c1ty) Максимальное количество золота   
[Задача №4.](#_y9wyx2jesy3p) Сувениры

[Задача №](#_p1ur54b6c1ty)5. Произведение матриц

# Задачи по варианту



**Код программы:**

import time

import tracemalloc

def collect\_signatures(segments):

# Сортируем по правому краю

segments.sort(key=lambda x: x[1])

points = []

last\_point = -1

for seg in segments:

if last\_point < seg[0]:

last\_point = seg[1]

points.append(last\_point)

return points

# Чтение входных данных

with open("input4.txt", "r") as f:

n = int(f.readline())

segments = [tuple(map(int, f.readline().split())) for \_ in range(n)]

# Запускаем замер памяти

tracemalloc.start()

# Засекаем время до вызова функции

start\_time = time.time()

# Вызываем функцию

points = collect\_signatures(segments)

# Засекаем время после вызова

end\_time = time.time()

# Получаем информацию о памяти

current, peak = tracemalloc.get\_traced\_memory()

tracemalloc.stop()

# Запись результата в файл

with open("output4.txt", "w") as f:

f.write(f"{len(points)}\n")

f.write(" ".join(map(str, points)))

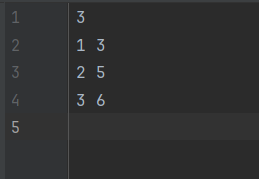
# Выводим время и память в консоль

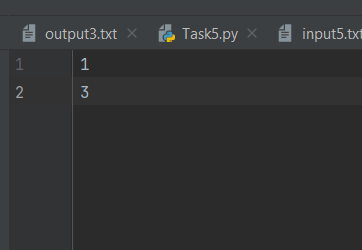
print(f"Время выполнения: {end\_time - start\_time:.6f} секунд")

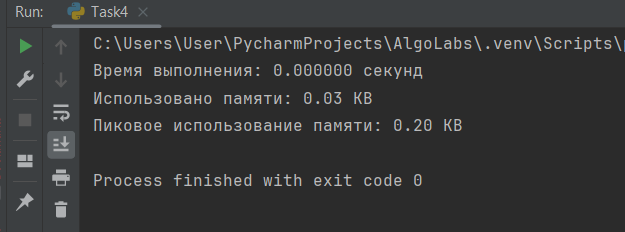
print(f"Использовано памяти: {current / 1024:.2f} KB")

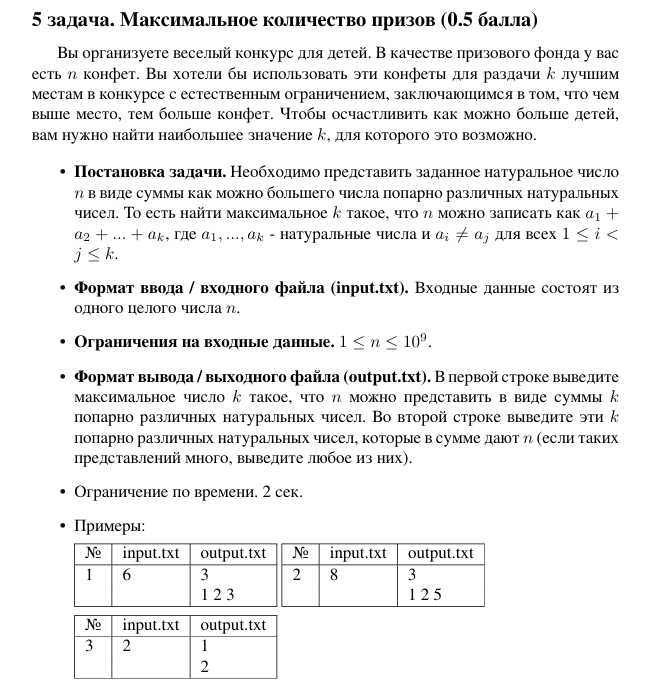
print(f"Пиковое использование памяти: {peak / 1024:.2f} KB")

Результат работы кода на примерах:









**Код программы:**

**import time**

**import tracemalloc**

**def max\_unique\_sum(n):**

**result = []**

**current = 1**

**while n > 0:**

**if n - current > current:**

**result.append(current)**

**n -= current**

**current += 1**

**else:**

**result.append(n)**

**break**

**return result**

**# Чтение входных данных**

**with open("input5.txt", "r") as f:**

**n = int(f.readline())**

**# Запускаем замер памяти**

**tracemalloc.start()**

**# Засекаем время до вызова функции**

**start\_time = time.time()**

**# Вызываем функцию**

**result = max\_unique\_sum(n)**

**# Засекаем время после вызова**

**end\_time = time.time()**

**# Получаем информацию о памяти**

**current, peak = tracemalloc.get\_traced\_memory()**

**tracemalloc.stop()**

**# Запись результата в файл**

**with open("output5.txt", "w") as f:**

**f.write(f"{len(result)}\n")**

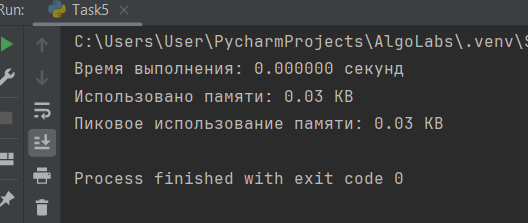
**f.write(" ".join(map(str, result)))**

**# Выводим время и память в консоль**

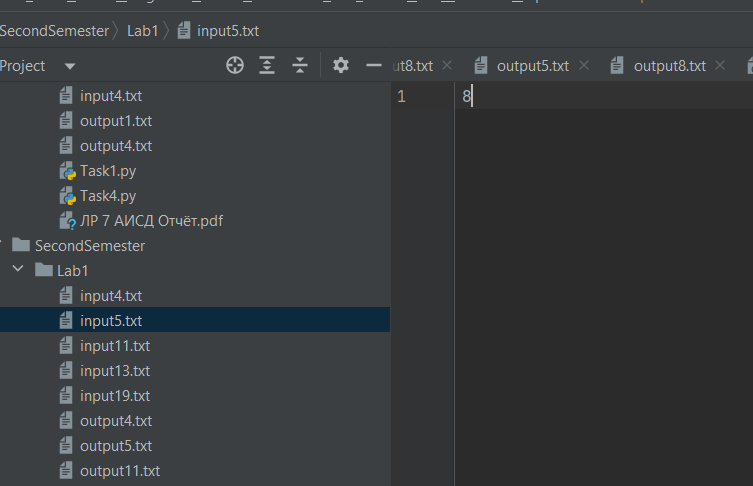
**print(f"Время выполнения: {end\_time - start\_time:.6f} секунд")**

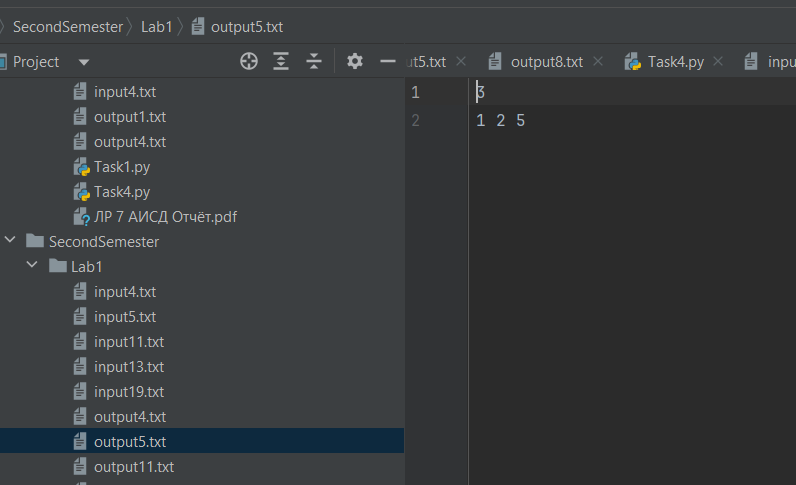
**print(f"Использовано памяти: {current / 1024:.2f} KB")**

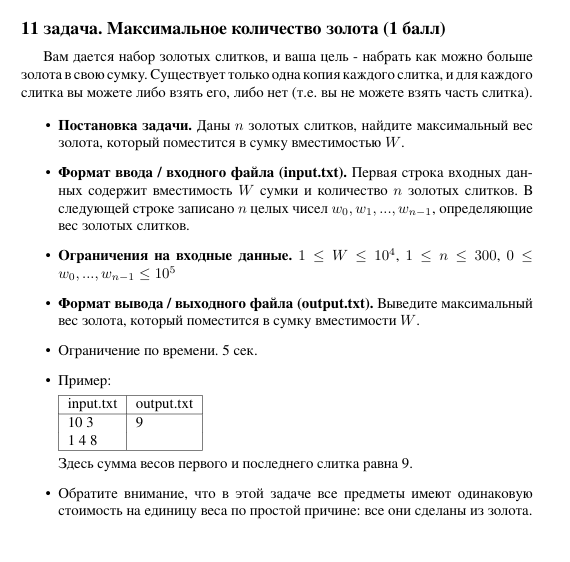
**print(f"Пиковое использование памяти: {peak / 1024:.2f} KB")**

****

Результат работы кода на примерах:

****

****

****

**Код программы:**

import time

import tracemalloc

def max\_gold(W, weights):

dp = [0] \* (W + 1)

for weight in weights:

for w in range(W, weight - 1, -1):

dp[w] = max(dp[w], dp[w - weight] + weight)

return dp[W]

# Чтение входных данных

with open("input11.txt", "r") as f:

W, n = map(int, f.readline().split())

weights = list(map(int, f.readline().split()))

# Замер памяти

tracemalloc.start()

# Засекаем время

start\_time = time.time()

# Вычисление результата

result = max\_gold(W, weights)

# Засекаем время

end\_time = time.time()

# Получаем информацию о памяти

current, peak = tracemalloc.get\_traced\_memory()

tracemalloc.stop()

# Запись результата

with open("output11.txt", "w") as f:

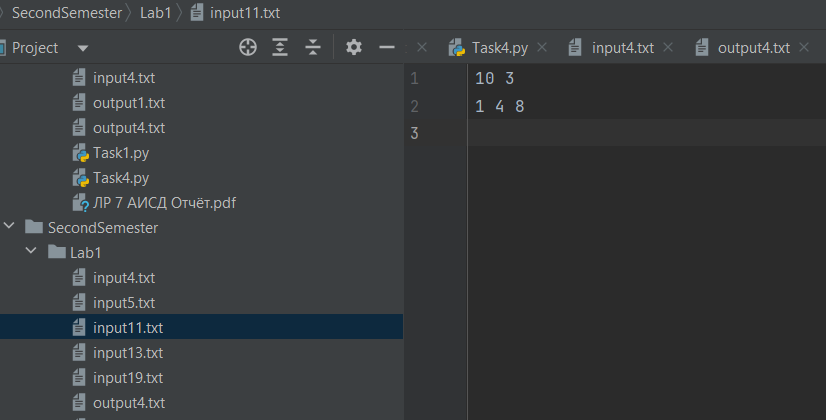
f.write(str(result))

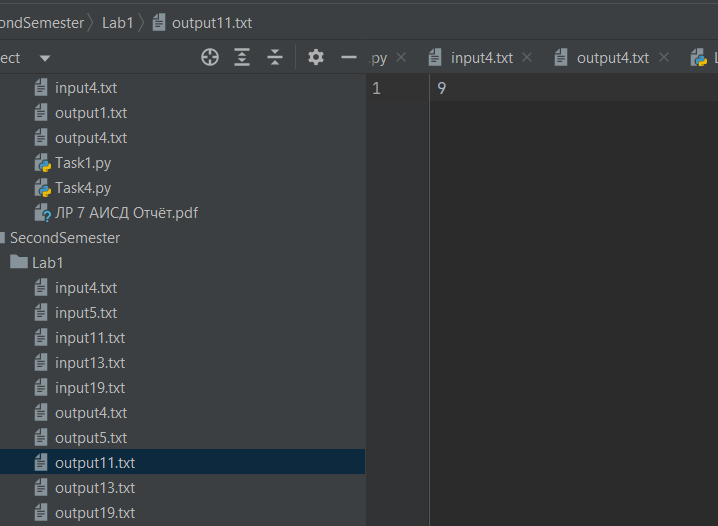
# Вывод в консоль

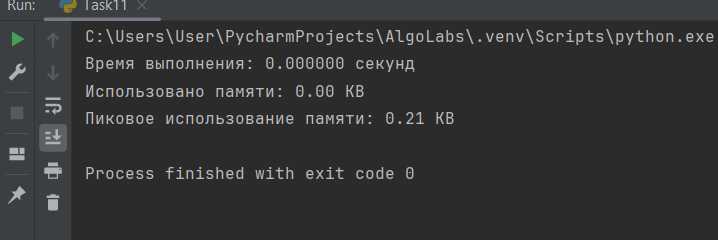
print(f"Время выполнения: {end\_time - start\_time:.6f} секунд")

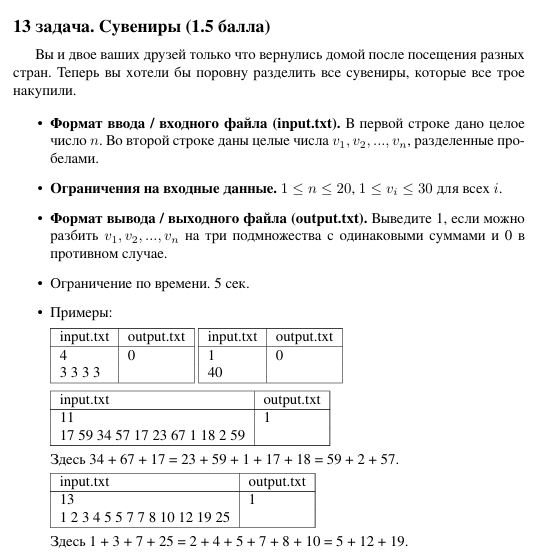
print(f"Использовано памяти: {current / 1024:.2f} KB")

print(f"Пиковое использование памяти: {peak / 1024:.2f} KB")



****

****

****

**Код программы:**

**import time**

**import tracemalloc**

**def can\_partition\_into\_three\_subsets(nums):**

**total\_sum = sum(nums)**

**if total\_sum % 3 != 0:**

**return 0**

**target = total\_sum // 3**

**n = len(nums)**

**dp = [[False] \* (target + 1) for \_ in range(n + 1)]**

**dp[0][0] = True**

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

**for j in range(target, -1, -1):**

**if j >= nums[i - 1]:**

**dp[i][j] = dp[i][j] or dp[i - 1][j - nums[i - 1]]**

**dp[i][j] = dp[i][j] or dp[i - 1][j]**

**return 1 if dp[n][target] else 0**

**# Чтение входных данных**

**with open('input13.txt', 'r') as file:**

**lines = file.readlines()**

**if len(lines) < 2:**

**raise ValueError("Файл input13.txt должен содержать как минимум две строки: количество чисел и сами числа.")**

**n = int(lines[0].strip())**

**nums = list(map(int, lines[1].strip().split()))**

**# Запуск замера времени и памяти**

**tracemalloc.start()**

**start\_time = time.perf\_counter()**

**# Основной вызов**

**result = can\_partition\_into\_three\_subsets(nums)**

**# Завершение замеров**

**end\_time = time.perf\_counter()**

**current, peak = tracemalloc.get\_traced\_memory()**

**tracemalloc.stop()**

**# Запись результата**

**with open('output13.txt', 'w') as file:**

**file.write(str(result))**

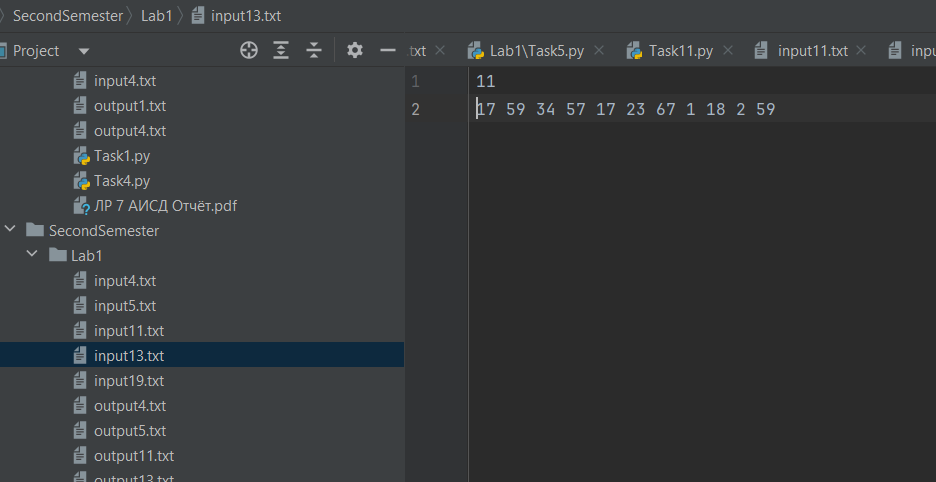
**# Вывод в консоль**

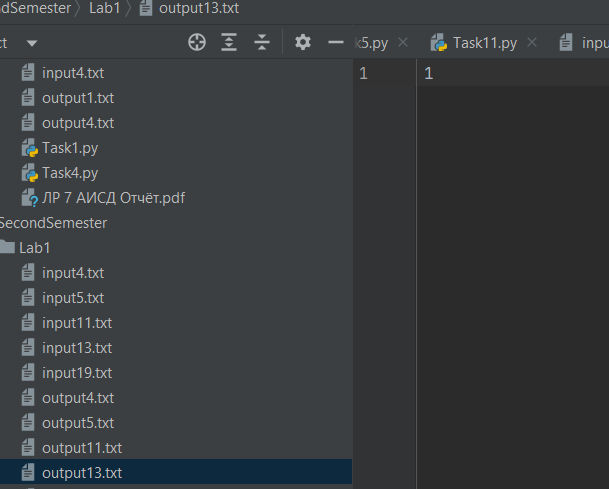
**print(f"Время выполнения: {end\_time - start\_time:.6f} секунд")**

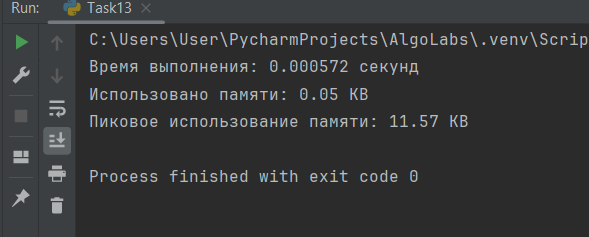
**print(f"Использовано памяти: {current / 1024:.2f} KB")**

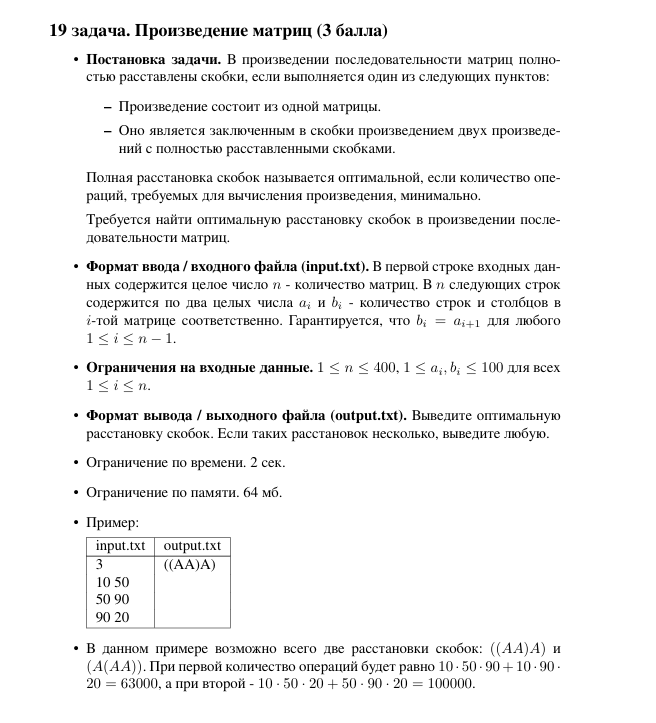
**print(f"Пиковое использование памяти: {peak / 1024:.2f} KB")**

Результат работы кода на примерах:

****

****

****

****

**Код программы:**

**import time**

**import tracemalloc**

**def matrix\_chain\_order(p):**

**n = len(p) - 1 # Количество матриц**

**dp = [[0]\*n for \_ in range(n)]**

**s = [[0]\*n for \_ in range(n)]**

**for l in range(2, n+1): # длина цепочки**

**for i in range(n - l + 1):**

**j = i + l - 1**

**dp[i][j] = float('inf')**

**for k in range(i, j):**

**cost = dp[i][k] + dp[k+1][j] + p[i]\*p[k+1]\*p[j+1]**

**if cost < dp[i][j]:**

**dp[i][j] = cost**

**s[i][j] = k**

**return dp, s**

**def build\_optimal\_parenthesization(s, i, j):**

**if i == j:**

**return f"A{i+1}"**

**else:**

**return "(" + build\_optimal\_parenthesization(s, i, s[i][j]) + \**

**build\_optimal\_parenthesization(s, s[i][j]+1, j) + ")"**

**# Чтение данных**

**with open("input19.txt", "r") as f:**

**n = int(f.readline())**

**dims = []**

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

**a, b = map(int, f.readline().split())**

**dims.append(a)**

**dims.append(b) # добавляем последний столбец последней матрицы**

**# Замер памяти и времени**

**tracemalloc.start()**

**start\_time = time.perf\_counter()**

**dp, s = matrix\_chain\_order(dims)**

**result = build\_optimal\_parenthesization(s, 0, n-1)**

**end\_time = time.perf\_counter()**

**current, peak = tracemalloc.get\_traced\_memory()**

**tracemalloc.stop()**

**# Запись результата**

**with open("output19.txt", "w") as f:**

**f.write(result)**

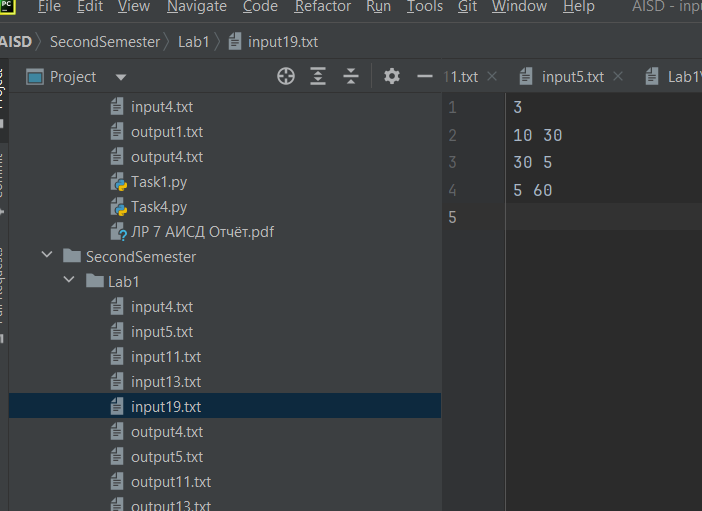
**# Вывод в консоль**

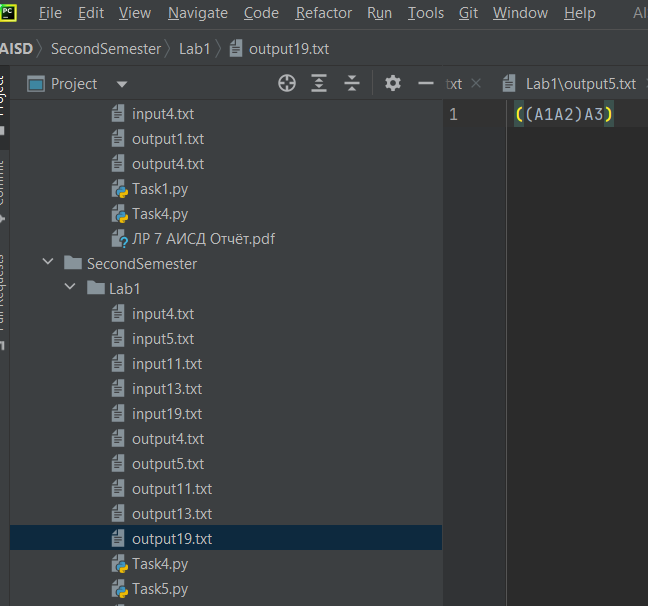
**print(f"Время выполнения: {end\_time - start\_time:.6f} секунд")**

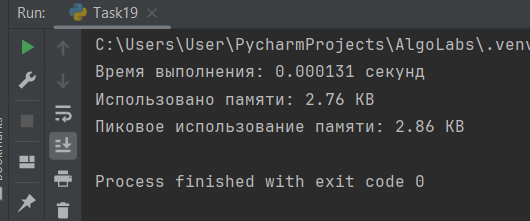
**print(f"Использовано памяти: {current / 1024:.2f} KB")**

**print(f"Пиковое использование памяти: {peak / 1024:.2f} KB")**

Результат работы кода на примерах:

****

****

****