САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №7
по курсу «Алгоритмы и структуры данных»
Тема: Динамическое программирование №1
Вариант 1

Выполнила:
Гайдук А. С.
К3241

Проверила:
Ромакина О. М.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

Задача №4. Наибольшая общая подпоследовательность двух последовательностей

Вычислить длину самой длинной общей подпоследовательности из двух последовательностей.

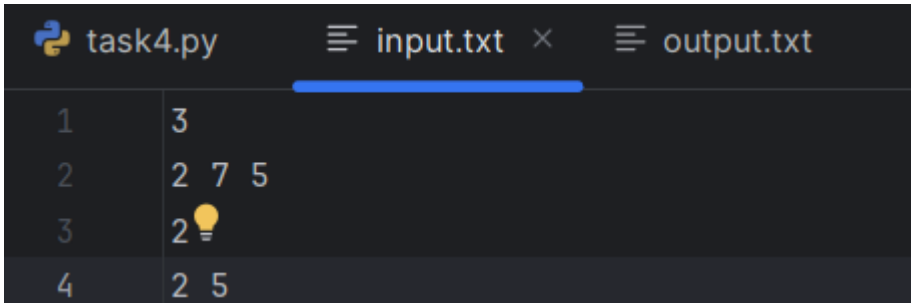
Даны две последовательности $A = (a_1, a_2, \dots, a_n)$ и $B = (b_1, b_2, \dots, b_m)$, найти длину их самой длинной общей подпоследовательности, т.е. наибольшее неотрицательное целое число p такое, что существуют индексы $1 \le i_1 < i_2 < \cdots < i_p \le n$ и $1 \le j_1 < j_2 < \cdots < j_p \le m$ такие, что $a_{i1} = b_{j1}, \dots, a_{ip} = b_{jp}$.

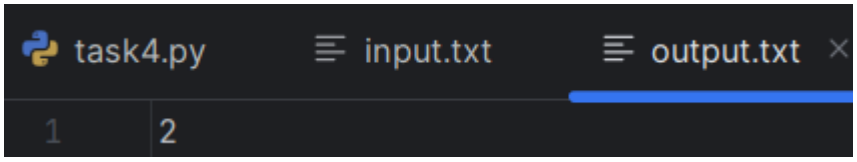
Листинг кода:

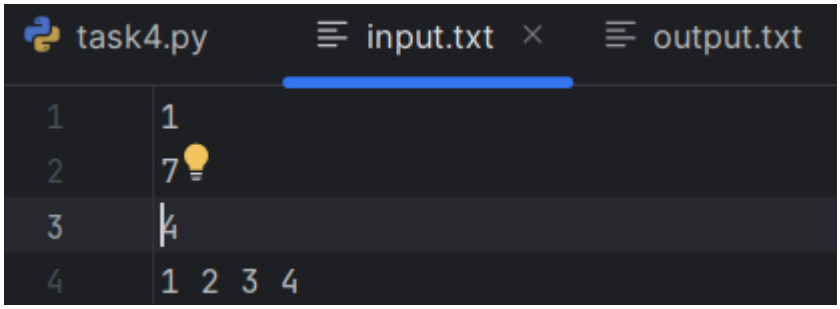
```
import tracemalloc
import time

tracemalloc.start()

def long_subsequence(n, A, m, B):
    table = [[0] * (m + 1) for _ in range(n + 1)]
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            if A[i - 1] == B[j - 1]:
                table[i][j] = table[i - 1][j - 1] + 1
            else:
                table[i][j] = max(table[i - 1][j], table[i][j - 1])

    return table[n][m]


with open('input.txt', 'r') as file:
    n = int(file.readline())
    A = list(map(int, file.readline().strip().split()))
    m = int(file.readline())
    B = list(map(int, file.readline().strip().split()))

start_time = time.perf_counter()

result = long_subsequence(n, A, m, B)

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()

with open('output.txt', 'w') as file:
    file.write(str(result))

print(f"Затраты памяти: {current / 10 ** 6:.6f} MB; Пиковое
использование: {peak / 10 ** 6:.6f} MB")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```

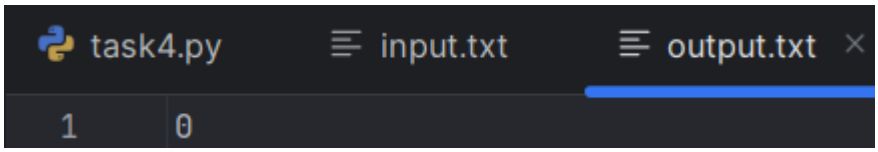
Текстовое объяснение решения:
Для решения я создала таблицу размером (n+1)(m+1) для хранения длины подпоследовательности. Если A, B совпадают, то длина увеличивается на единицу от предыдущей подпоследовательности, иначе берется максимум между значением, полученным без текущего элемента из первой и второй последовательностей, чтобы не пропустить возможные совпадения.

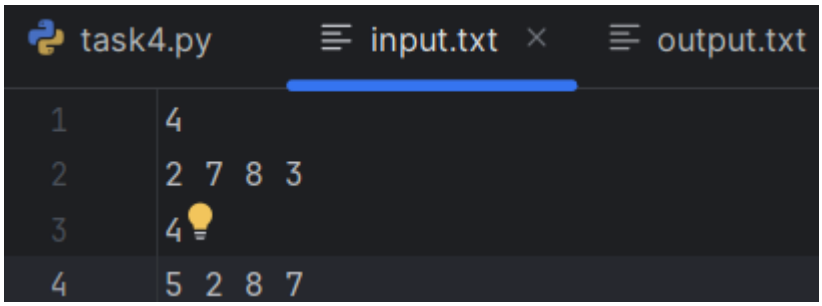
Результат работы кода на примерах из текста задачи:

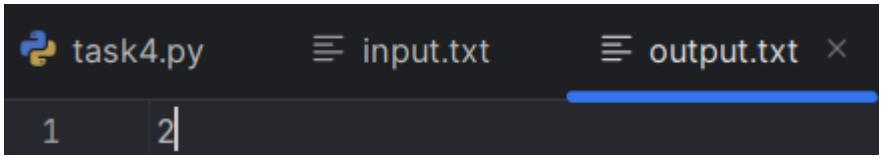




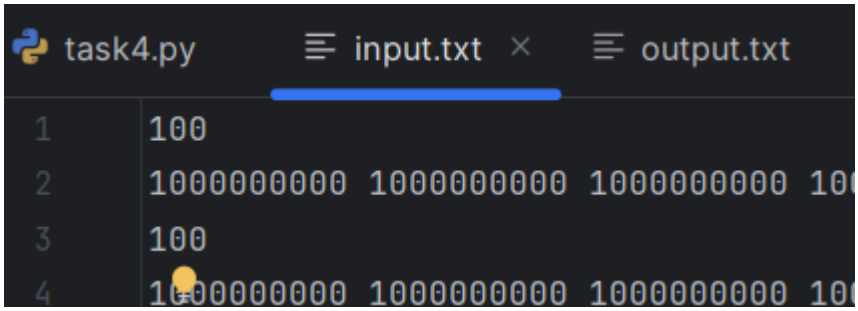


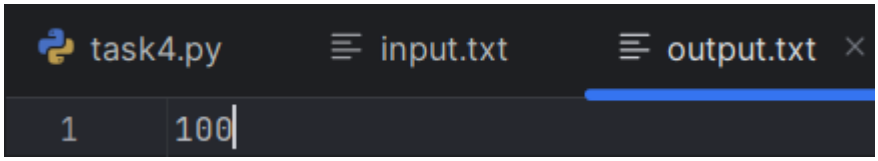


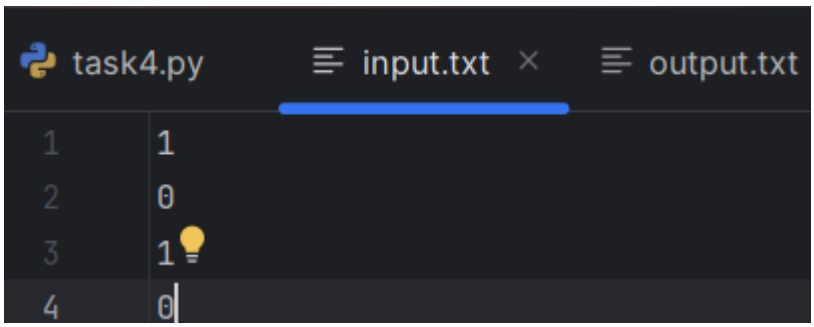


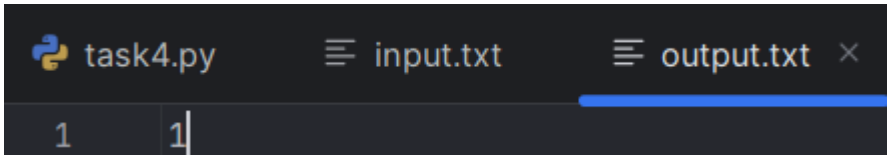


Результат работы кода на максимальных и минимальных значениях:









	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000020 сек	0.037667 MB
Пример из задачи	0.000031 сек	0.037749 MB
Пример из задачи	0.000021 сек	0.037721 MB
Пример из задачи	0.000082 сек	0.037759 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.001448 сек	0.120977 MB

Вывод по задаче:

В ходе выполнения данной задачи у меня получилось реализовать алгоритм поиска наибольшей подпоследовательности между двумя последовательностями.

Задача №5. Наибольшая общая подпоследовательность трех последовательностей

Вычислить длину самой длинной общей подпоследовательности из трех последовательностей.

Даны три последовательности $A = (a_1, a_2, \dots, a_n)$, $B = (b_1, b_2, \dots, b_m)$ и $C = (c_1, c_2, \dots, c_l)$, найти длину их самой длинной общей подпоследовательности, т.е. наибольшее неотрицательное целое число p такое, что существуют индексы $1 \le i_1 < i_2 < \dots < i_p \le n$, $1 \le j_1 < j_2 < \dots < j_p \le m$ и $1 \le k_1 < k_2 < \dots < k_p \le l$ такие, что $a_{i1} = b_{j1} = c_{k1}, \dots, a_{ip} = b_{jp} = c_{kp}$

Листинг кода:

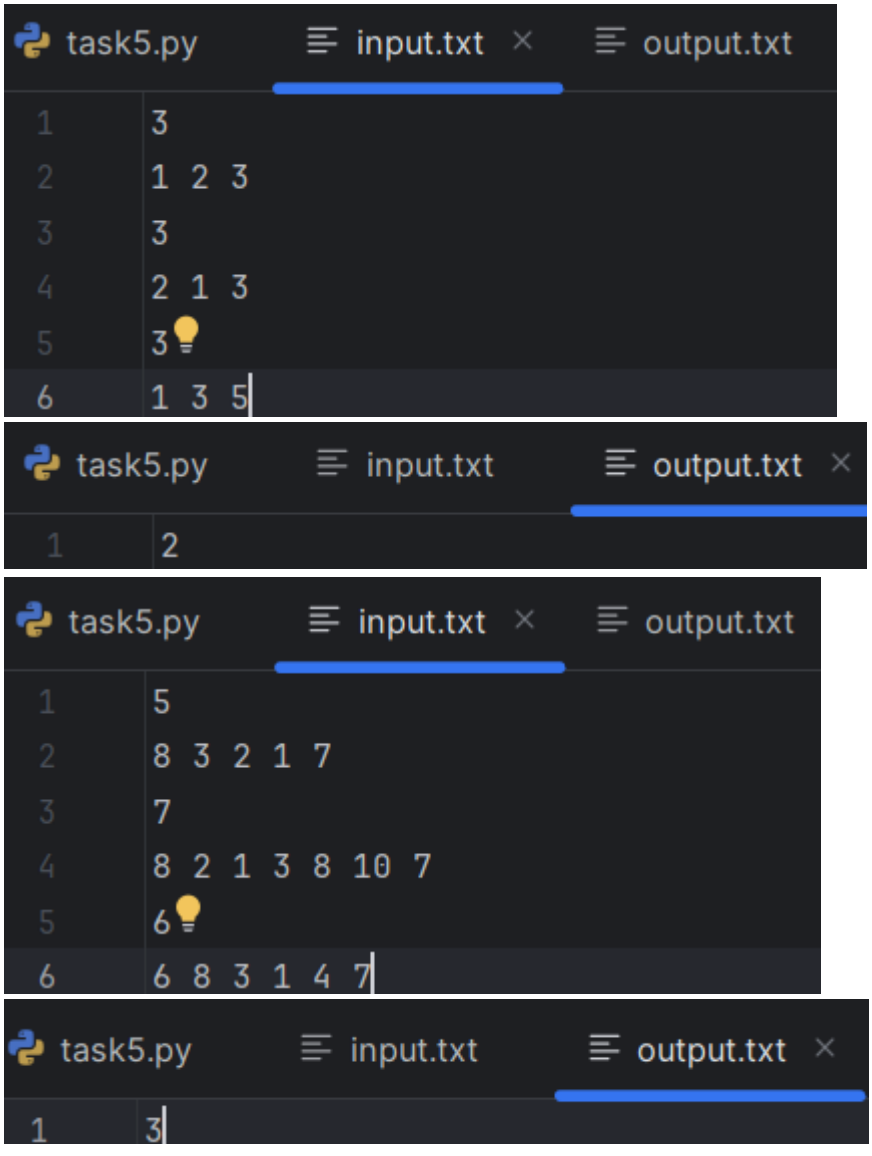
```
import tracemalloc
import time

tracemalloc.start()

def long_subsequence(n, A, m, B, l, C):
    table = [[[0] * (l + 1) for _ in range(m + 1)] for __ in range(n+1)]
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            for k in range(1, l + 1):
                if A[i - 1] == B[j - 1] == C[k - 1]:
                    table[i][j][k] = table[i - 1][j - 1][k - 1] + 1
                else:
                    table[i][j][k] = max(table[i - 1][j][k], table[i][j - 
1][k], table[i][j][k - 1])

    return table[n][m][l]


with open('input.txt', 'r') as file:
    n = int(file.readline())
    A = list(map(int, file.readline().strip().split()))
    m = int(file.readline())
    B = list(map(int, file.readline().strip().split()))
    l = int(file.readline())
    C = list(map(int, file.readline().strip().split()))

start_time = time.perf_counter()

result = long_subsequence(n, A, m, B, l, C)
```

```
end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()

with open('output.txt', 'w') as file:
    file.write(str(result))

print(f"Затраты памяти: {current / 10 ** 6:.6f} MB; Пиковое
использование: {peak / 10 ** 6:.6f} MB")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```

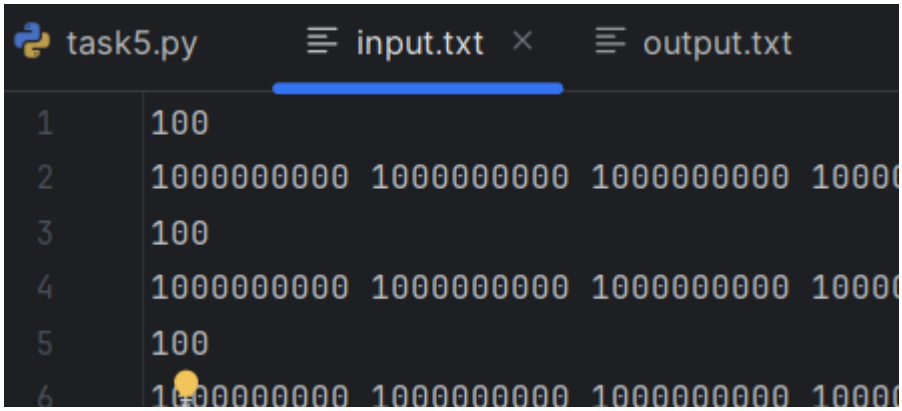
Текстовое объяснение решения:

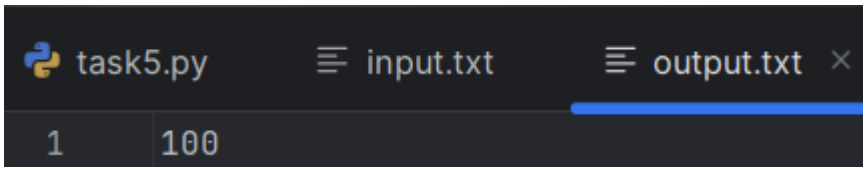
Для решения этой задачи я преобразовала код, написанный к задаче №4, создав трехмерный массив table и прописав состояния для совпадения и несовпадения элементов трех последовательностей.

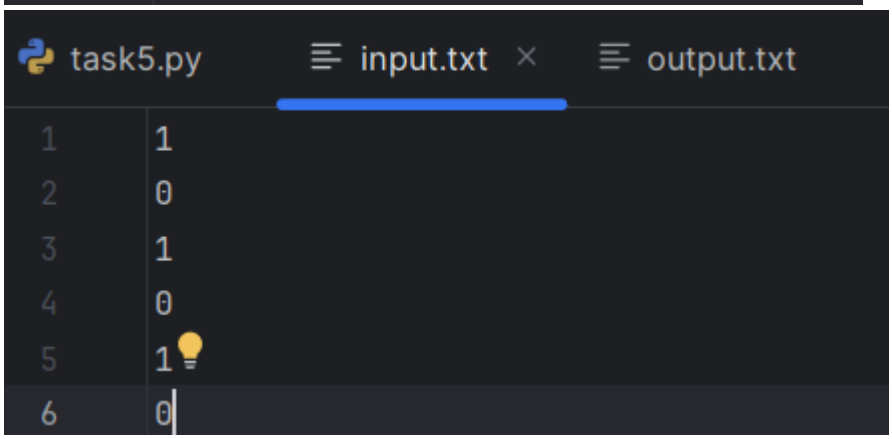
Результат работы кода на примерах из текста задачи:

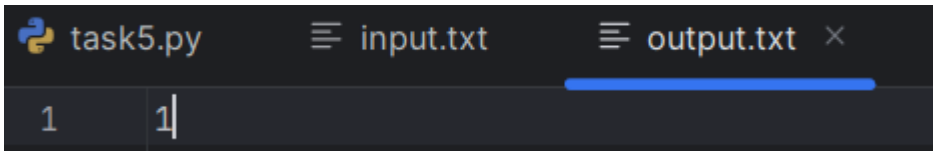


Результат работы кода на максимальных и минимальных значениях:









	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000039 сек	0.037761 MB
Пример из задачи	0.000079 сек	0.037883 MB
Пример из задачи	0.000348 сек	0.037908 MB
Верхняя граница диапазона значений входных данных из	0.243933 сек	8.946659 MB

текста задачи		

Вывод по задаче:

В ходе выполнения данной задачи у меня получилось преобразовать алгоритм предыдущей задачи так, чтобы можно было найти наибольшую подпоследовательность между тремя последовательностями.

Дополнительные задачи

Задача №1. Обмен монет

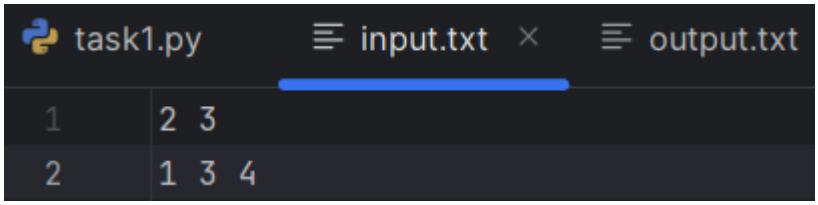
Как мы уже поняли из лекции, не всегда «жадное» решение задачи на обмен монет работает корректно для разных наборов номиналов монет. Например, если доступны номиналы 1, 3 и 4, жадный алгоритм поменяет 6 центов, используя три монеты (4 + 1 + 1), в то время как его можно изменить, используя всего две монеты (3 + 3). Теперь ваша цель – применить динамическое программирование для решения задачи про обмен монет для разных номиналов.

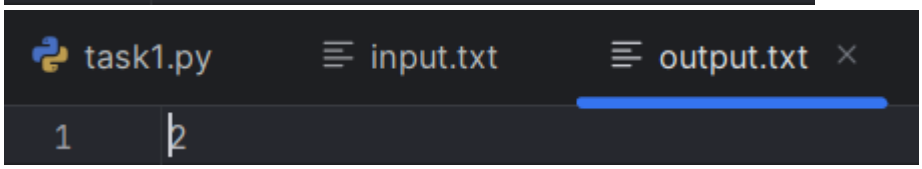
Листинг кода:

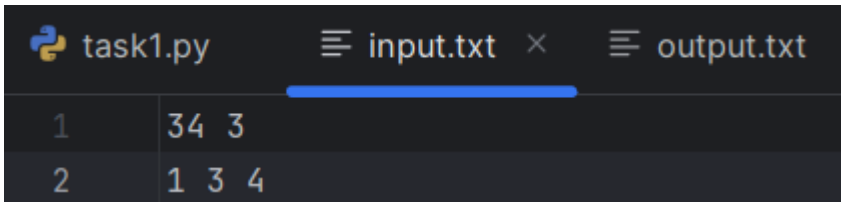
```
import tracemalloc
import time

tracemalloc.start()

def exchanger(money, coins):
    MAX_VALUE = money + 1
    dp = [MAX_VALUE] * (money + 1)
    dp[0] = 0
    for coin in coins:
        for j in range(coin, money + 1):
            dp[j] = min(dp[j], dp[j - coin] + 1)

    return dp[money] if dp[money] != MAX_VALUE else -1




with open('input.txt', 'r') as file:
    money, k = map(int, file.readline().strip().split())
    coins = list(map(int, file.readline().strip().split()))

start_time = time.perf_counter()

result = exchanger(money, coins)

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()

with open('output.txt', 'w') as file:
    file.write(str(result))

print(f"Затраты памяти: {current / 10 ** 6:.6f} MB; Пиковое
использование: {peak / 10 ** 6:.6f} MB")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```

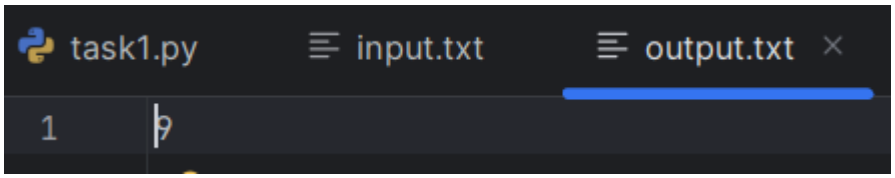
Текстовое объяснение решения:
Значение MAX_VALUE, равное money + 1, используется как недостижимая сумма (т. к. монет для размена денег не может быть больше, чем самих денег). Я создала массив длиной money + 1, установив базовый случай для суммы 0. Перебирая каждую монету из списка, проверяем какие суммы можно составить с её номинала. Далее я обновляю dp, выбирая минимальное значение между текущим количеством монет для размена суммы j и количеством монет для суммы j – coin плюс одна дополнительная монета coin.

Результат работы кода на примерах из текста задачи:

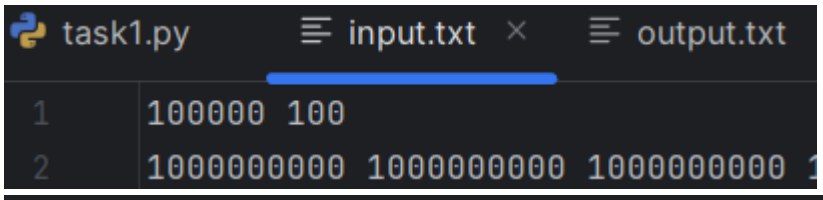


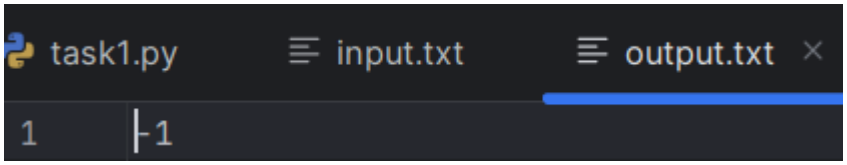


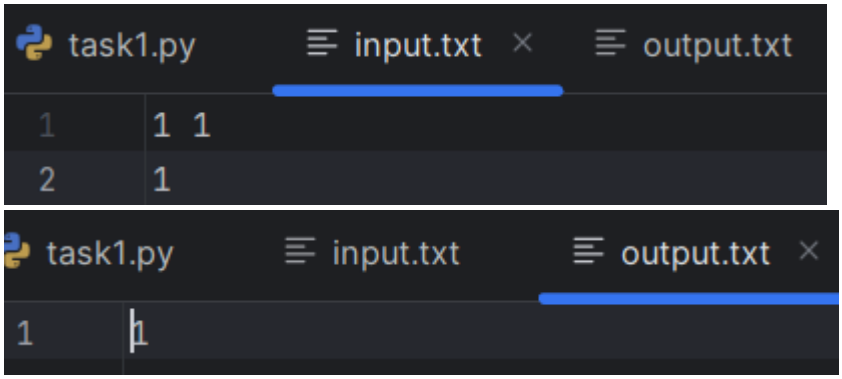




Результат работы кода на максимальных и минимальных значениях:







	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000011 сек	0.037575 MB
Пример из задачи	0.000014 сек	0.037625 MB
Пример из задачи	0.000072 сек	0.037626 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.000675 сек	0.827321 MB

Вывод по задаче:
В ходе выполнения этой задачи я научилась реализовывать обменник монет с помощью динамического программирования.

Задача №6. Наибольшая возрастающая подпоследовательность

Дана последовательность, требуется найти ее наибольшую возрастающую подпоследовательность.

Листинг кода:

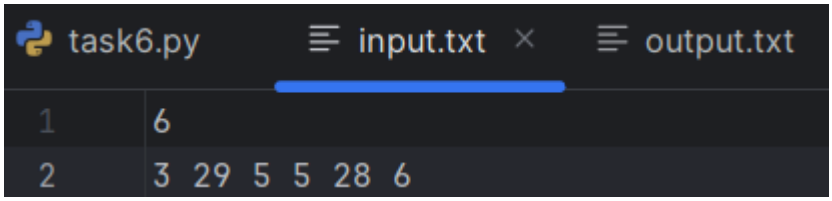
```
import tracemalloc
import time

tracemalloc.start()

def max_increasing_seq(n, A):
    table = [1] * n
    parent = [-1] * n
```

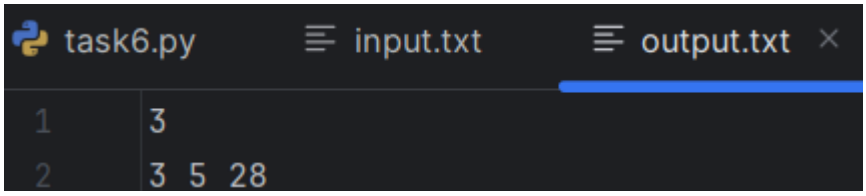
```
    for i in range(1, n):
        for j in range(i):
            if A[i] > A[j] and table[i] < table[j] + 1:
                table[i] = table[j] + 1
                parent[i] = j
    max_length = max(table)
    max_index = table.index(max_length)
    result = []
    while max_index != -1:
        result.append(A[max_index])
        max_index = parent[max_index]
    result.reverse()
    return max_length, result


with open('input.txt', 'r') as file:
    n = int(file.readline())
    A = list(map(int, file.readline().strip().split()))

start_time = time.perf_counter()

length, max_sequence = max_increasing_seq(n, A)

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()



with open('output.txt', 'w') as file:
    file.write(str(length) + '\n')
    file.write(' '.join(map(str, max_sequence)))


print(f"Затраты памяти: {current / 10 ** 6:.6f} MB; Пиковое
использование: {peak / 10 ** 6:.6f} MB")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```

Текстовое объяснение решения:

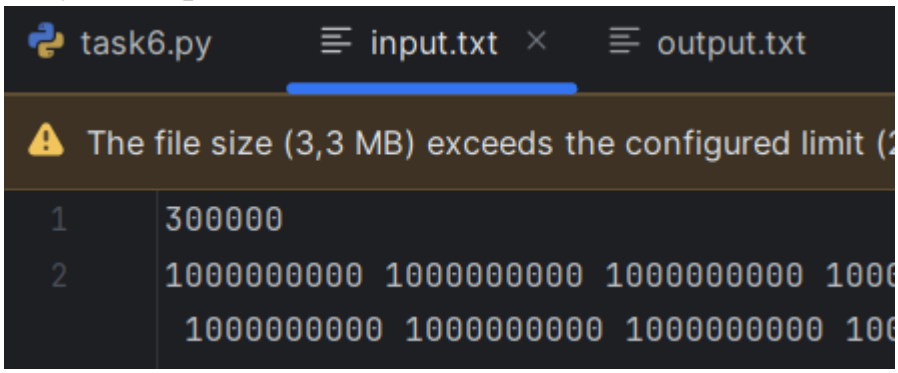
Я создала список table ля хранения максимальной длины, изначально для каждого элемента это 1. Также я создала список parent для хранения индекса предыдущего элемента A[i]. Для каждого элемента A[i] проверяю все элементы A[j], которые стоят перед ним, и если текущий элемент A[i] больше предыдущего A[j], то A[i] можно добавить к подпоследовательности, заканчивающейся на A[j]. Если после добавления длина стала больше текущей, то увеличиваем длину table и в parent указываем, что A[i] идет после A[j] в подпоследовательности. Далее я иду назад по массиву parent, начиная с последнего элемента подпоследовательности, и добавляю элементы в результат, а потом переворачиваю его через reverse().

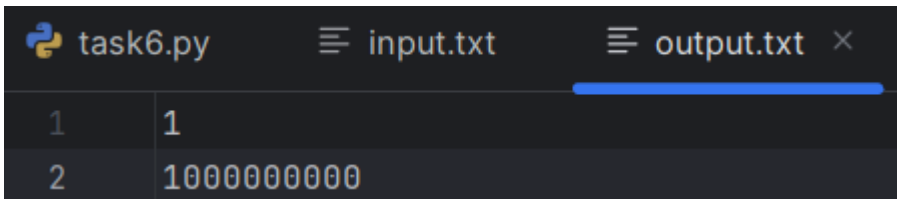
Результат работы кода на примерах из текста задачи:

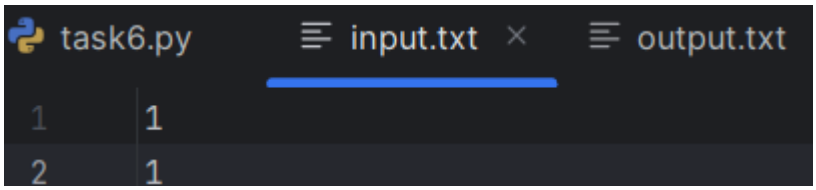


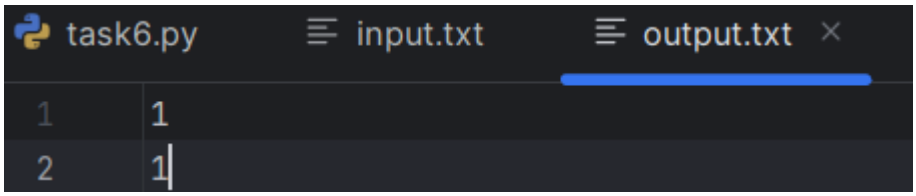


Результат работы кода на максимальных и минимальных значениях:









	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000026 сек	0.037525 MB

Пример из задачи	0.000029 сек	0.037591 MB
Верхняя граница диапазона значений входных данных из текста задачи	1.148316 сек	30.150410 MB

Вывод по задаче:
Я научилась находить наибольшую возрастающую подпоследовательность с помощью динамического программирования.

Вывод

В ходе выполнения данной лабораторной работы я ознакомилась с принципами динамического программирования.