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

Отчет по лабораторной работе №4

по курсу «Алгоритмы и структуры данных»

Тема: Подстроки

Вариант 1

Выполнила:
Гайдук А. С.
К3241

Проверила:
Ромакина О. М.

Санкт-Петербург
2025 г.

Содержание отчета

Задачи по варианту

Задача №1. Наивный поиск подстроки в строке [2 s, 256 Mb, 1 балл]

Даны строки p и t. Требуется найти все вхождения строки p в строку t в качестве подстроки.

- **Формат ввода / входного файла (input.txt).** Первая строка входного файла содержит p, вторая – t. Строки состоят из букв латинского алфавита.
- **Ограничения на входные данные.** $1 \le |p|, |t| \le 10^4$.
- **Формат вывода / выходного файла (output.txt).** В первой строке выведите число вхождений строки p в строку t. Во второй строке выведите в возрастающем порядке номера символов строки t, с которых начинаются вхождения p. Символы нумеруются с единицы.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

Листинг кода:

```
import time
import tracemalloc

tracemalloc.start()

def compute_z_function(s):
    n = len(s)
    z = [0] * n
    l, r = 0, 0

    for i in range(1, n):
        if i <= r:
            z[i] = min(r - i + 1, z[i - l])
        while i + z[i] < n and s[z[i]] == s[i + z[i]]:
            z[i] += 1
        if i + z[i] - 1 > r:
            l, r = i, i + z[i] - 1
    return z

def z_search(p, t):
    s = p + '$' + t
    z = compute_z_function(s)
    len_p = len(p)
    occurrences = []
    for i in range(len_p + 1, len(s)):
        if z[i] == len_p:
            occurrences.append(i - len_p)
    return occurrences


with open('input.txt', 'r') as file:
    p = file.readline().strip()
    t = file.readline().strip()

start_time = time.perf_counter()

result = z_search(p, t)

end_time = time.perf_counter()
```

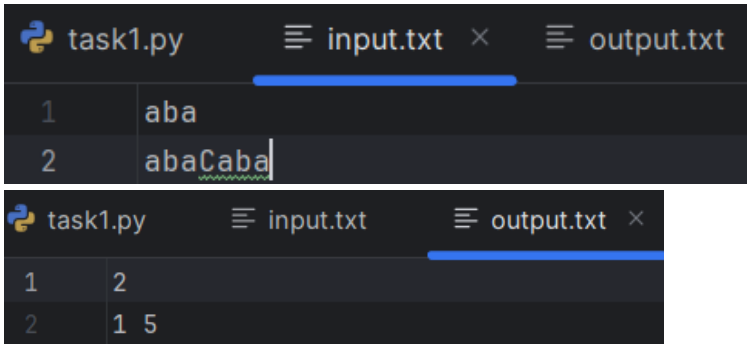
```
current, peak = tracemalloc.get_traced_memory()


with open('output.txt', 'w') as file:
    file.write(f"{len(result)}\n")
    if result:
        file.write(" ".join(map(str, result)) + "\n")

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

Текстовое объяснение решения:

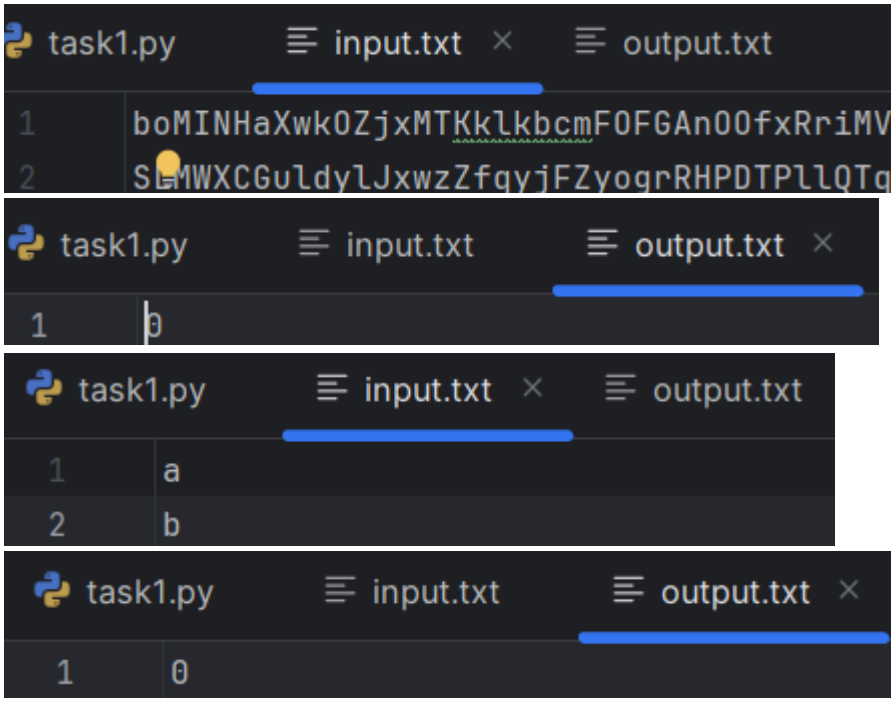
Для решения этой задачи я использовала Z-функцию. Функция compute_z_function вычисляет Z-функцию для строки s. Я создала список z, который хранит значения z-функции для каждой позиции строки, а также переменные l, r, которые хранят левую и правую границы текущего отрезка, на котором Z-функция уже вычислена. Затем я прохожу по всем символам строки, начиная с позиции 1. Если текущая позиция находится внутри отрезка [l, r], то значение z-функции определяется как минимум длины оставшейся части отрезка и значения Z-функции для симметричной позиции. Пока символы строки s совпадают на позициях z[i] и i + z[i], увеличиваем значение Z-функции для i на единицу. Если текущий отрезок [i, i + z[i] – 1] выходит за пределы [l, r], обновляем границы l и r. Возвращаю список z.

Функция z_search ищет все вхождения подстроки p в строку t с использованием Z-функции. Я создаю новую строку s, которая состоит из подстроки p, символа-разделителя и строки t. Затем вызываю функцию compute_z_function для строки s. Далее я создаю список для хранения позиций вхождений p в t и прохожусь по всем позициям строки s, начиная с символа после разделителя. Если значение Z-функции для позиции i равно длине подстроки p, то подстрока p начинается с позиции (i – len_p) строки t. Позицию добавляю в список для результата.

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	0.000013 сек	0.037685 MB
Пример из задачи	0.000033 сек	0.037785 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.073355 сек	0.228452MB

Вывод по задаче:

Я смогла найти все вхождения подстроки в строку с использованием Z-функции.

Задача №3. Паттерн в тексте [2 s, 256 Mb, 1 балл]

В этой задаче ваша цель – реализовать алгоритм Рабина-Карпа для поиска заданного шаблона (паттерна) в заданном тексте.

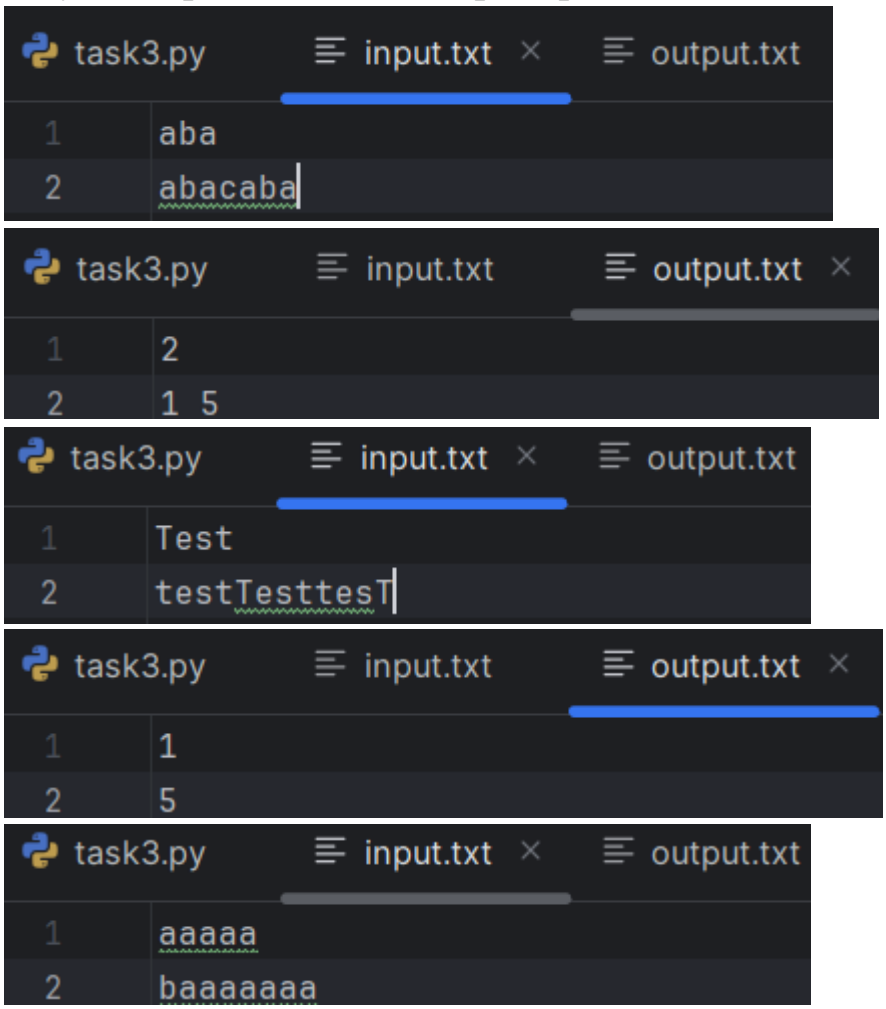
- **Формат ввода / входного файла (input.txt).** На входе две строки: паттерн P и текст T. Требуется найти все вхождения строки P в строку T в качестве подстроки.
- **Ограничения на входные данные.** $1 \leq |P|, |T| \leq 10^6$. Паттерн и текст содержат только латинские буквы.
- **Формат вывода / выходного файла (output.txt).** В первой строке выведите число вхождений строки P в строку T. Во второй строке выведите в возрастающем порядке номера символов строки T, с которых начинаются вхождения P. Символы нумеруются с единицы.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

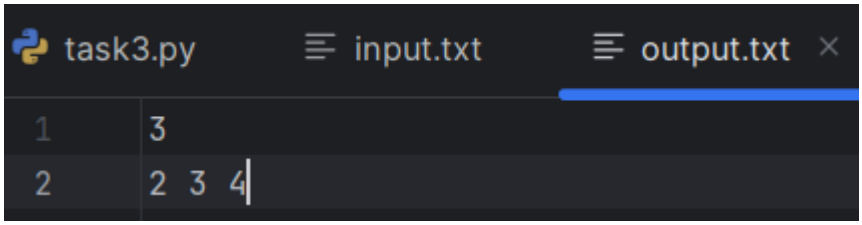
Листинг кода:

```
import time
import tracemalloc

tracemalloc.start()

def rabin_karp(pattern, text):
    pattern_len = len(pattern)
    text_len = len(text)
    if text_len < pattern_len:
        return []
    result = []
    pattern_hash = hash(pattern)
    for i in range(0, text_len - pattern_len + 1):
        subtext = text[i:i + pattern_len]
        if hash(subtext) == pattern_hash and subtext == pattern:
            result.append(i + 1)
    return result


with open('input.txt', 'r') as f:
    pattern = f.readline().strip()
    s = f.readline().strip()

start_time = time.perf_counter()

result = rabin_karp(pattern, s)

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

with open('output.txt', 'w') as f:
    f.write(str(len(result)) + '\n')
    f.write(' '.join(map(str, result)))

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

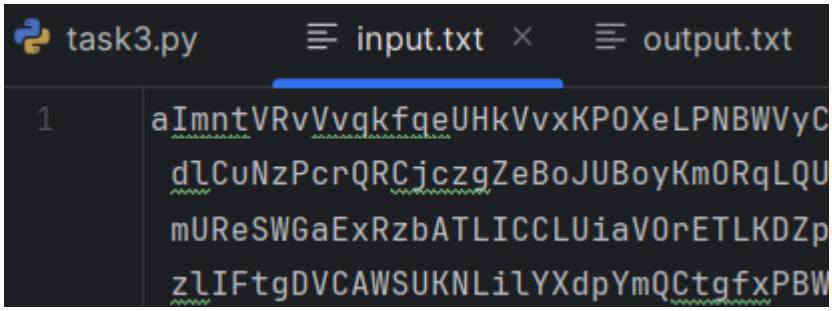
Текстовое объяснение решения:
Я реализовала функцию rabin_karp, которая принимает подстроку p (pattern) и текст, в котором ищется подстрока. Далее я вычисляю длины подстроки и строки и проверяю, что длина текста больше или равна длине подстроки (иначе возвращаю пустой список). Далее я создаю список для хранения позиций вхождений подстроки в строку.
С использованием встроенной функции hash() я вычисляю хеш подстроки. Циклом прохожусь по всем возможным начальным позициям подстроки в строке. Для каждой позиции извлекаю подстроку T[i:i + len(p)]. Если хеш данной подстроки совпадает с хешом подстроки p, то выполняю точное сравнение строк с помощью «==». Если строки совпадают, то добавляю позицию i + 1 (нумерация с 1) в результат.

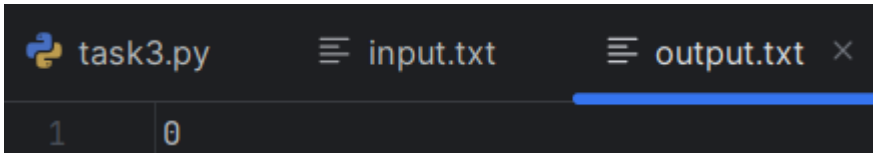
Результат работы кода на примерах из текста задачи:

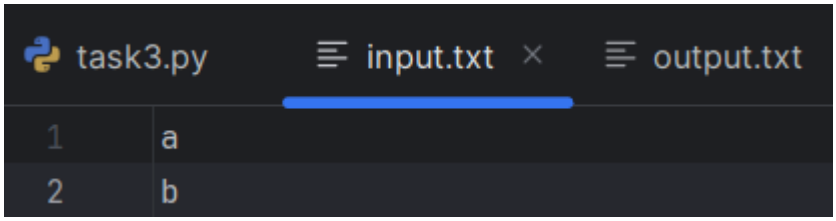


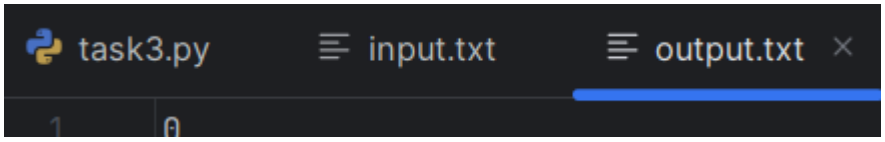


Результат работы кода на максимальных и минимальных значениях:









	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000007 сек	0.037525 MB
Пример из задачи	0.000012 сек	0.037625 MB
Пример из задачи	0.000017 сек	0.037637 MB
Пример из задачи	0.000011 сек	0.037631 MB
Верхняя граница диапазона значений входных данных из	0.000360 сек	3.036231 MB

текста задачи		

Вывод по задаче:

Я реализовала алгоритм Рабина-карпа для поиска заданного паттерна в заданном тексте.

Задача №9. Декомпозиция строки [15 s, 512 Mb, 2 балла]

Строка `ABCABCDEDEDEF` содержит подстроку `ABC`, повторяющуюся два раза подряд, и подстроку `DE`, повторяющуюся три раза подряд. Таким образом, ее можно записать как `ABC*2+DE*3+F`, что занимает меньше места, чем исходная запись той же строки.

Ваша задача – построить наиболее экономное представление данной строки s в виде, продемонстрированном выше, а именно, подобрать такие $s_1, a_1, ..., s_k, a_k$, где s_i - строки, а a_i - числа, чтобы $s = s_1 \cdot a_1 + ... + s_k \cdot a_k$. Под операцией умножения строки на целое положительное число подразумевается конкатенация одной или нескольких копий строки, число которых равно числовому множителю, то есть, `ABC*2=ABCABC`. При этом требуется минимизировать общую длину итогового описания, в котором компоненты разделяются знаком `+`, а умножение строки на число записывается как умножаемая строка и множитель, разделенные знаком `*`. Если же множитель равен единице, его, вместе со знаком `*`, допускается не указывать.

- **Формат ввода / входного файла (input.txt).** Одна строка входного файла содержит s. Строка состоит из букв латинского алфавита.
- **Ограничения на входные данные.** $1 \le |s| \le 5 \cdot 10^3$.
- **Формат вывода / выходного файла (output.txt).** Выведите оптимальное представление строки, данной во входном файле. Если оптимальных представлений несколько, выведите любое.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

Листинг кода:

```
import time
import tracemalloc

tracemalloc.start()

def decomposition(s):
    n = len(s)
    dp = [float('inf')] * (n + 1)
    dp[0] = 0
    parts = [''] * (n + 1)

    for i in range(n + 1):
        if dp[i] == float('inf'):
            continue
        for j in range(i + 1, n + 1):
            substring = s[i:j]
            length = j - i
            count = 1
            k = j
            while k + length <= n and s[k:k+length] == substring:
                k += length
                count += 1
            representation = f"{substring}*{count}" if count > 1 else
substring
            if dp[i] + len(representation) < dp[k]:
```

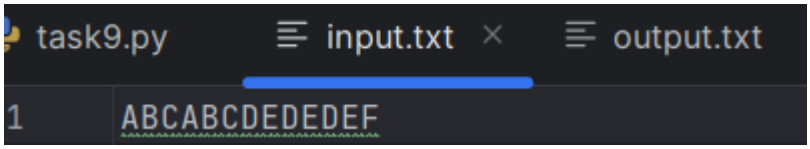
```
                dp[k] = dp[i] + len(representation)
                parts[k] = (parts[i] + ('+' if parts[i] else '') +
representation) if parts[i] else representation

    return parts[n]

with open('input.txt', 'r') as file:
    s = file.readline().strip()

start_time = time.perf_counter()

result = decomposition(s)

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

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


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

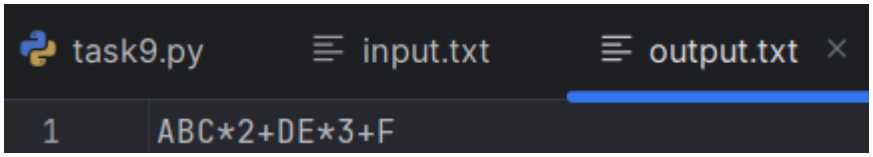
Текстовое объяснение решения:

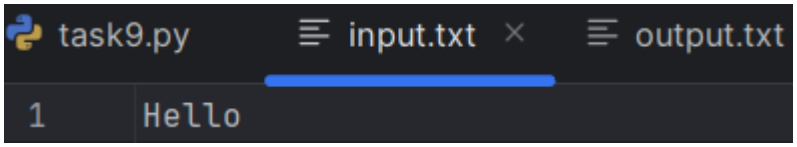
Я создала функцию decomposition, которая принимает на вход строку и возвращает её оптимальное представление. Я создала список ДП длины n+1, где каждый элемент изначально равен бесконечности, он хранит минимальную длину представления подстроки. Затем я создала массив parts, где каждый элемент хранит оптимальное представление подстроки. Прохожусь по всем возможным начальным позициям подстрок и, если текущее значение dp[i] равно бесконечности, пропускаю, т. к. до этой позиции невозможно добраться. Далее я перебираю все возможные конечные позиции подстрок, начиная с i + 1. Извлекаю подстроку от i до j, определяю её длину и инициализирую счетчик повторений подстроки. Начинаю проверку с позиции j: пока подстрока повторяется, увеличиваю счетчик и сдвигаю позицию k.

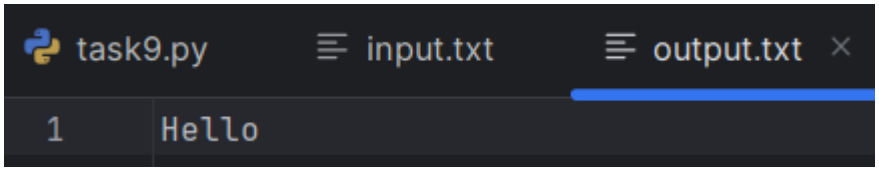
Далее формирую строку представления, добавляя множитель, если нужно. Если новое представление (представление до i + текущее представление строки s[i...k] короче текущего, обновляю ДП и parts. Обновляю минимальную длину представления для позиции k, а также оптимальное представление для позиции k.

Результат работы кода на примерах из текста задачи:

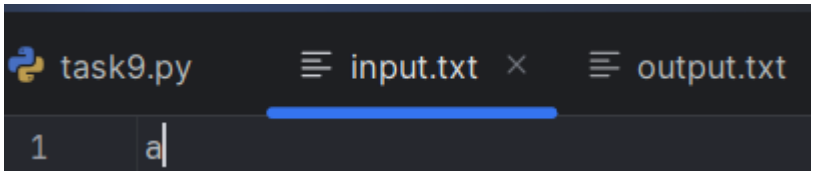


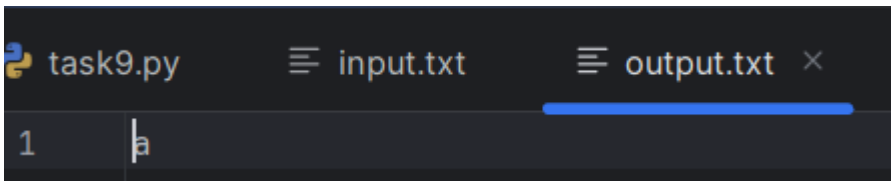






Результат работы кода на максимальных и минимальных значениях:





	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000013 сек	0.037522 MB
Пример из задачи	0.000180 сек	0.037588 MB
Пример из задачи	0.000031 сек	0.037572 MB

Вывод по задаче:

Я реализовала алгоритм для построения оптимального представления строки с помощью динамического программирования.

Дополнительные задачи

Задача №2. Карта [2 s, 256 Mb, 1 балл]

В далеком 1744 году во время долгого плавания в руки капитана Александра Смоллетта попала древняя карта с указанием местонахождения сокровищ. Однако расшифровать ее содержание было не так уж и просто.

Команда Александра Смоллетта догадалась, что сокровища находятся на x шагов восточнее красного креста, однако определить значение числа она не смогла. По возвращению на материк Александр Смоллетт решил обратиться за помощью в расшифровке послания к знакомому мудрецу. Мудрец поведал, что данное послание таит за собой некоторое число. Для вычисления этого числа необходимо было удалить все пробелы между словами, а потом посчитать количество способов вычеркнуть все буквы кроме трех так, чтобы полученное слово из трех букв одинаково читалось слева направо и справа налево.

Александр Смоллетт догадывался, что число, зашифрованное в послании, и есть число x. Однако, вычислить это число у него не получилось.

После смерти капитана карта была безнадежно утеряна до тех пор, пока не оказалась в ваших руках. Вы уже знаете все секреты, осталось только вычислить число x.

- **Формат ввода / входного файла (input.txt).** В единственной строке входного файла дано послание, написанное на карте.
- **Ограничения на входные данные.** Длина послания не превышает $3 \cdot 10^5$. Гарантируется, что послание может содержать только строчные буквы английского алфавита и пробелы. Также гарантируется, что послание не пусто. Послание не может начинаться с пробела или заканчиваться им.
- **Формат вывода / выходного файла (output.txt).** Выведите одно число x – число способов вычеркнуть из послания все буквы кроме трех так, чтобы оставшееся слово одинаково читалось слева направо и справа налево.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

Листинг кода:

```
import time
import tracemalloc

tracemalloc.start()

def palindromics_count(message):
    message = message.replace(' ', '')
    n = len(message)
    prefix_count = [0] * 26
    suffix_count = [0] * 26
    for char in message:
        suffix_count[ord(char) - ord('a')] += 1
    total_triplets = 0
    for i in range(n):
        current_char = message[i]
        suffix_count[ord(current_char) - ord('a')] -= 1

        if i > 0:
            for j in range(26):
                total_triplets += prefix_count[j] * suffix_count[j]

        prefix_count[ord(current_char) - ord('a')] += 1
    return total_triplets



with open('input.txt', 'r') as file:
    message = file.readline().strip()
```

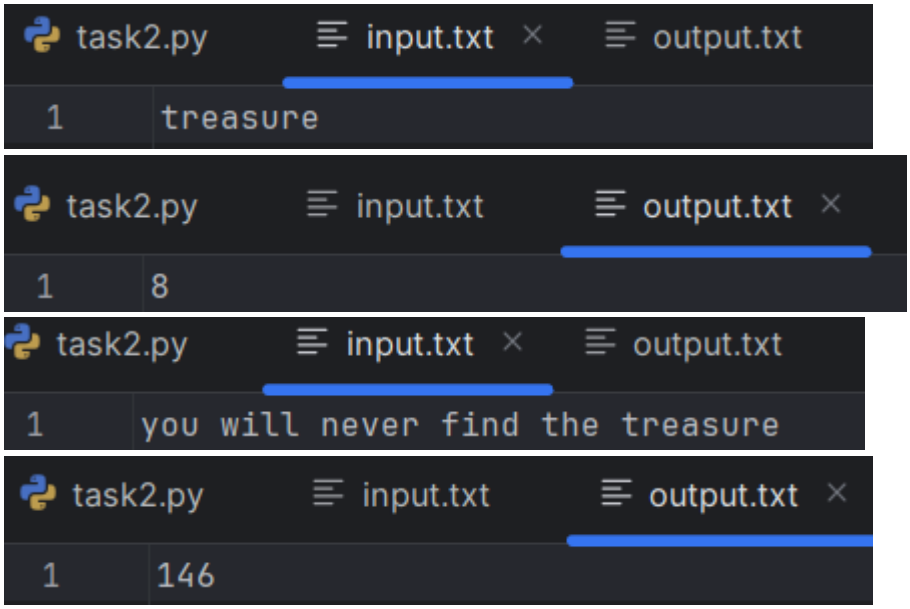
```
start_time = time.perf_counter()

result = palindromics_count(message)

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

Текстовое объяснение решения:

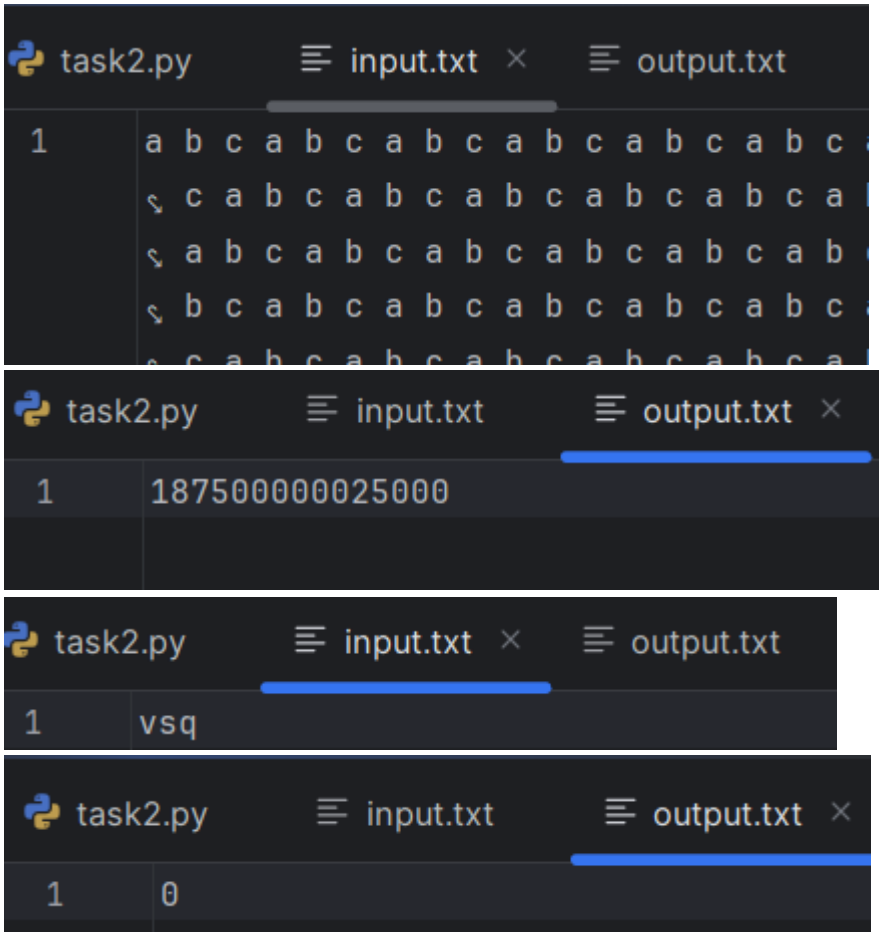
Я создала функцию для подсчета палиндромов с вычеркиванием. Сначала я удаляю все пробелы из строки и вычисляю длину строки без пробелов. Далее создаю два списка, изначально по 26 нулей. Прохожусь по каждому символу сообщения и увеличиваю счетчик в suffix_count. Создаю переменную для хранения общего количества палиндромных троек и прохожу по каждому символу строки. Для текущего символа уменьшаем его счетчик в suffix_count, т. к. он больше не находится справа от текущей позиции.

Если текущая позиция не первая, прохожу по всем 26 буквам и добавляю к счетчику троек произведение соответствующих счетчиков из suffix_count и prefix_count. После обработки символа увеличиваю его счетчик в prefix_count, т. к. он теперь слева от следующей позиции.

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	0.000021 сек	0.037568 MB
Пример из задачи	0.000021 сек	0.037578 MB
Пример из задачи	0.000053 сек	0.037626 MB
Верхняя граница диапазона значений входных данных из текста задачи	1.633680 сек	0.630769 MB

Вывод по задаче:

Я реализовала алгоритм для определения количества способов вычеркнуть из строки все буквы кроме трех так, чтобы образовать палиндром.

Вывод

В ходе выполнения данной лабораторной работы я изучила работу с подстроками, алгоритм Рабина-Карпа и Z-функции, а также применила свои знания на практике.