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 20

Выполнил:

Смирнов Георгий Валерьевич

К3139

Проверила:

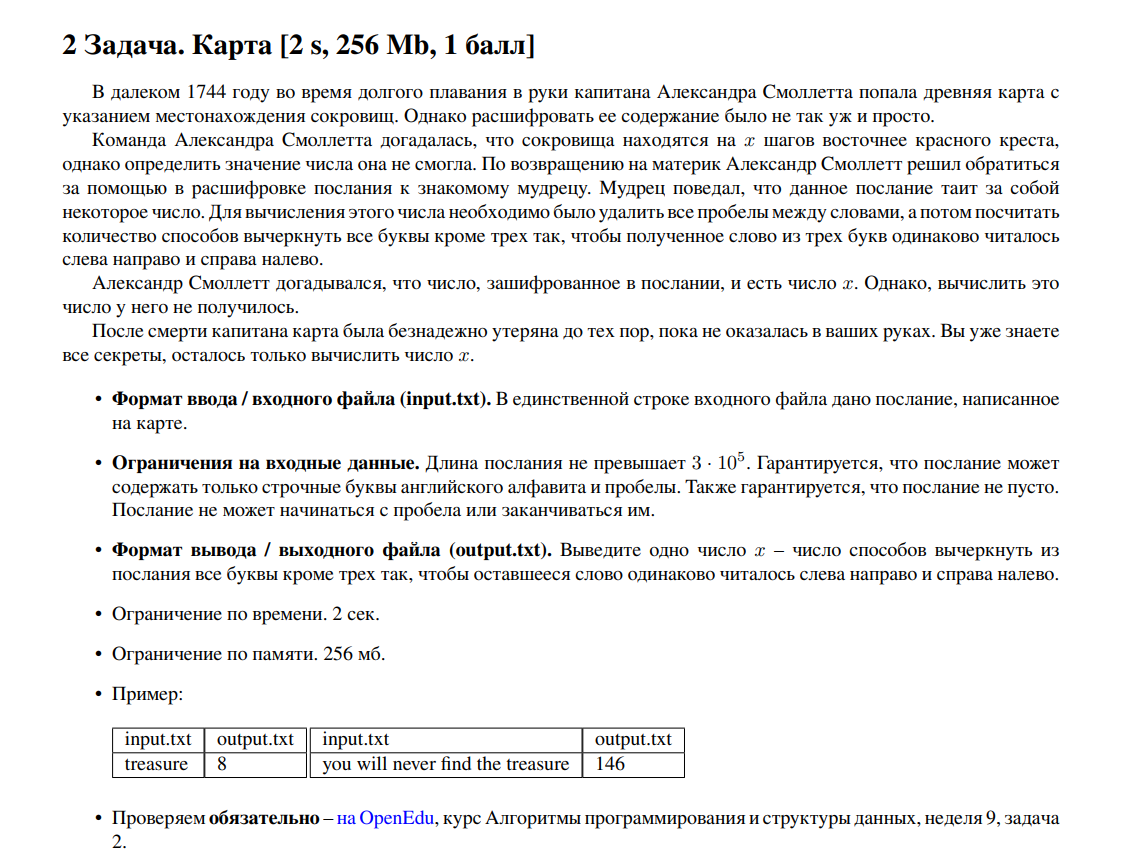
Афанасьев А.В.

Санкт-Петербург

2024 г.

# Задачи по варианту

## Задача №2.



**Код программы**

import time

import tracemalloc

def count\_palindromic\_triplets(message):

# Удаляем все пробелы из сообщения

message = message.replace(" ", "")

n = len(message)

count = 0

# Перебираем все возможные комбинации трех букв

for i in range(n):

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

for k in range(j + 1, n):

if message[i] == message[k]:

count += 1

return count

# Чтение входного файла

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

message = file.readline().strip()

# Начало измерения времени и памяти

start\_time = time.time()

tracemalloc.start()

# Вычисление количества палиндромных триплетов

result = count\_palindromic\_triplets(message)

# Окончание измерения времени и памяти

end\_time = time.time()

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

tracemalloc.stop()

# Вывод результата в консоль

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

print(f"Использование памяти: {current / 10\*\*6} MB (текущая), {peak / 10\*\*6} MB (пиковая)")

# Запись результата в выходной файл

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

file.write(f"{result}\n")

**Текстовое объяснение решения**

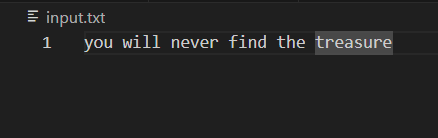
Программа читает входное сообщение из файла input.txt и удаляет из него все пробелы.

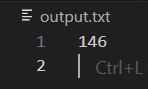
Затем она перебирает все возможные комбинации трех букв в строке и проверяет, образуют ли они палиндром (первый и третий символы должны быть одинаковыми).

Если комбинация образует палиндром, счетчик увеличивается.

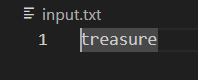
Результат работы кода на примерах из текста задачи:

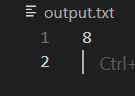
1)





2)

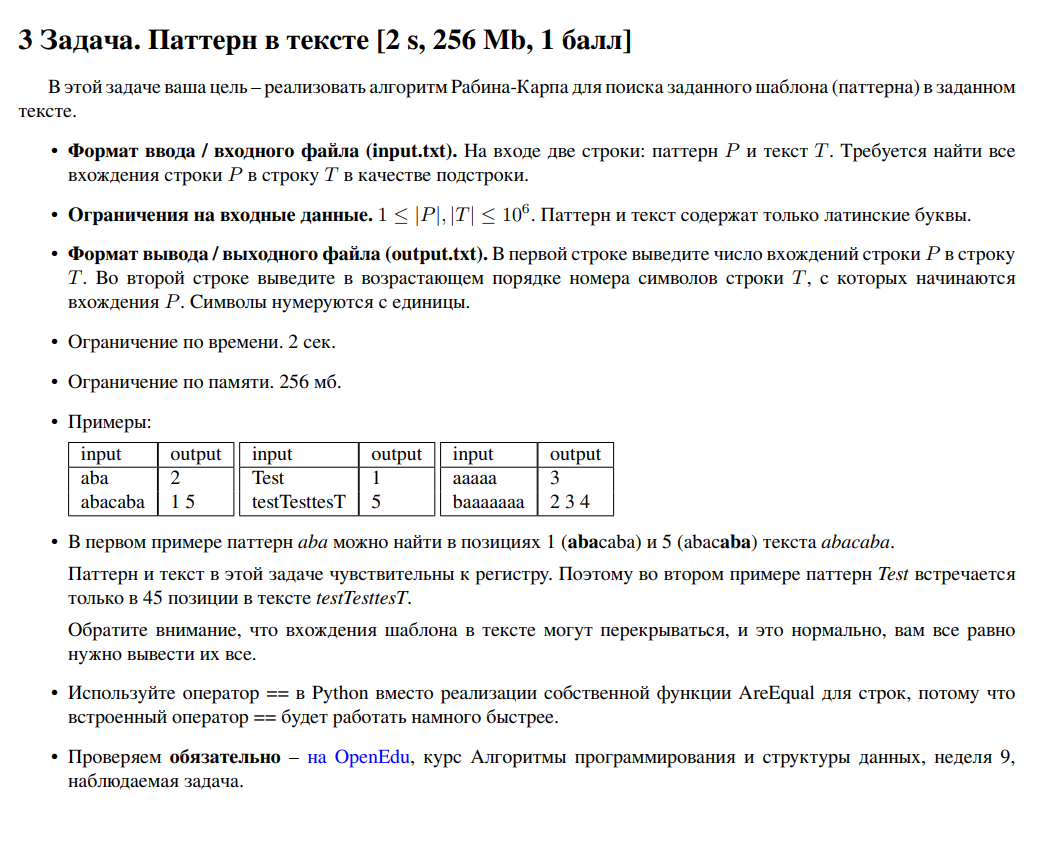




|  | Время выполнения, с | Затраты памяти, МБ |
| --- | --- | --- |
| Пример из задачи | 0.001616477966308593 | 0.000212 |
| Пример из задачи | 0.0000824000 | 0.000144 |

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти

## Задача №3.



**Код программы**

def rabin\_karp(pattern, text):

p\_len = len(pattern)

t\_len = len(text)

p\_hash = hash(pattern)

result = []

for i in range(t\_len - p\_len + 1):

t\_sub\_hash = hash(text[i:i + p\_len])

if p\_hash == t\_sub\_hash and text[i:i + p\_len] == pattern:

result.append(i + 1) # Нумерация с 1

return result

def main():

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

pattern = file.readline().strip()

text = file.readline().strip()

positions = rabin\_karp(pattern, text)

count = len(positions)

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

file.write(f"{count}\n")

file.write(" ".join(map(str, positions)) + "\n")

if \_\_name\_\_ == "\_\_main\_\_":

import time

import tracemalloc

start\_time = time.time()

tracemalloc.start()

main()

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

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

print(f"Использование памяти: {current / 10\*\*6} MB; Пиковое использование памяти: {peak / 10\*\*6} MB")

tracemalloc.stop()

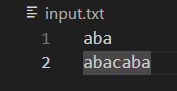
**Текстовое объяснение решения**

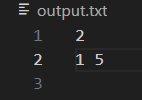
Для поиска всех вхождений шаблона в текст используется вспомогательная функция rabin\_karp, которая принимает шаблон и текст. В этой функции инициализируется длина шаблона и текста, а также вычисляется хеш-код шаблона. Создается пустой список для хранения результатов. Затем перебираются все возможные подстроки текста длиной, равной длине шаблона, и для каждой подстроки вычисляется хеш-код. Если хеш-код подстроки совпадает с хеш-кодом шаблона и сама подстрока совпадает с шаблоном, то индекс начала этой подстроки добавляется в список результатов. В конце функция возвращает список всех позиций вхождений шаблона в текст.

Функция main открывает файл input.txt и считывает первую строку как шаблон и вторую строку как текст. Затем вызывает функцию rabin\_karp для поиска всех вхождений шаблона в текст и получает список позиций вхождений. Записывает количество вхождений и сами позиции в файл output.txt.

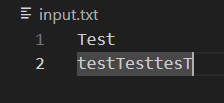
Результат работы кода на примерах из текста задачи:

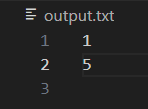
1)





2)

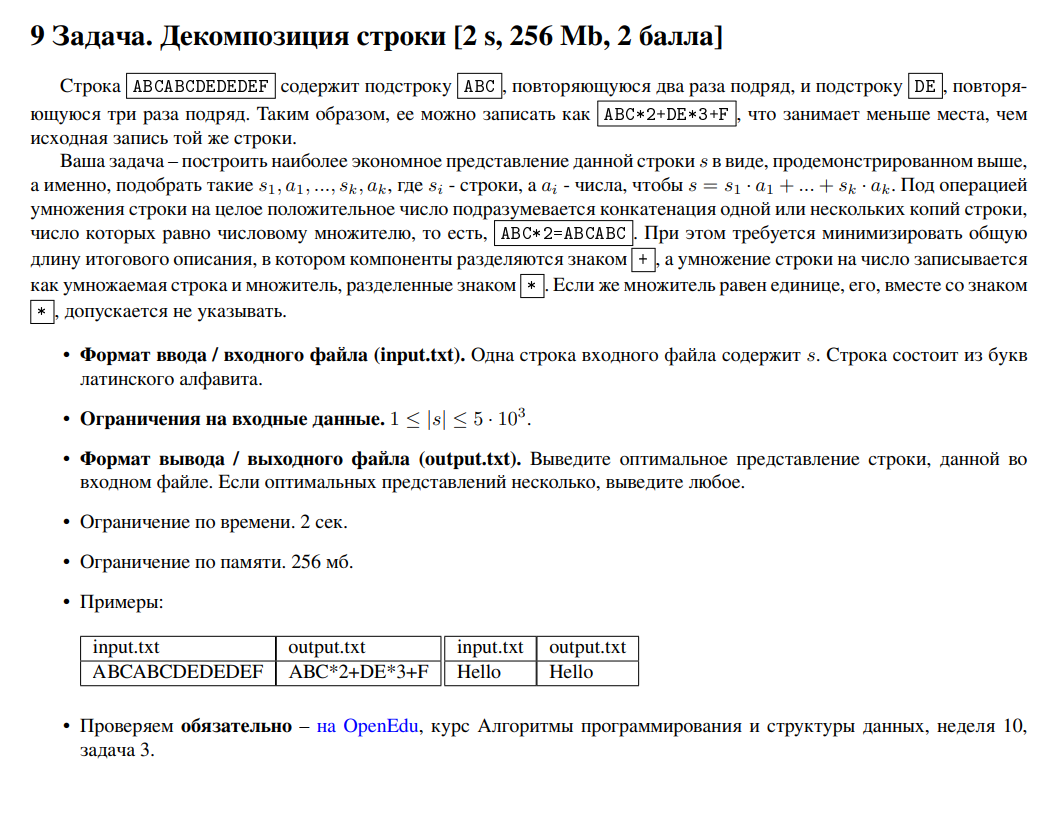




|  | Время выполнения, с | Затраты памяти, МБ |
| --- | --- | --- |
| Пример из задачи | 0.001010656356811523 | 0.018031 |
|  | 0.001014471054077148 | 0.018043 |

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти

## Задача №9.



**Код программы**

def encode\_string(s):

n = len(s)

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

for l in range(n):

for i in range(n - l):

j = i + l

substr = s[i:j + 1]

dp[i][j] = substr

if j - i < 4:

continue

for k in range(i, j):

if len(dp[i][j]) > len(dp[i][k] + '+' + dp[k + 1][j]):

dp[i][j] = dp[i][k] + '+' + dp[k + 1][j]

for k in range(1, len(substr)):

repeat\_str = substr[:k]

if repeat\_str and len(substr) % len(repeat\_str) == 0 and substr == repeat\_str \* (len(substr) // len(repeat\_str)):

encoded = f"{dp[i][i + k - 1]}\*{len(substr) // len(repeat\_str)}"

if len(encoded) < len(dp[i][j]):

dp[i][j] = encoded

return dp[0][n - 1]

def main():

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

s = file.readline().strip()

result = encode\_string(s)

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

file.write(result + '\n')

if \_\_name\_\_ == "\_\_main\_\_":

import time

import tracemalloc

start\_time = time.time()

tracemalloc.start()

main()

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

print(f"Время выполнения: {time.time() - start\_time:.10f} секунд")

print(f"Использование памяти: {current / 10\*\*6:.6f} MB; Пиковое использование памяти: {peak / 10\*\*6:.6f} MB")

tracemalloc.stop()

**Текстовое объяснение решения**

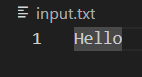
Для декомпозиции строки используется вспомогательная функция encode\_string, которая принимает строку s. В этой функции инициализируется двумерный список dp для хранения оптимальных представлений подстрок. Затем перебираются все возможные подстроки строки s и для каждой подстроки вычисляется ее оптимальное представление.

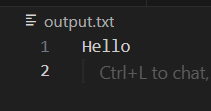
Если длина подстроки меньше 4, она сохраняется в исходном виде. Для подстрок длиной 4 и более проверяются все возможные разбиения на две части, и если объединение этих частей дает более короткое представление, чем текущее, оно обновляется. Также проверяются все возможные повторяющиеся подстроки, и если подстрока может быть представлена как повторение более короткой подстроки, это представление сохраняется, если оно короче текущего.

В конце функция возвращает оптимальное представление всей строки s.

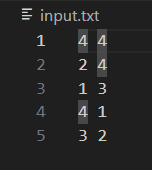
Результат работы кода на примерах из текста задачи:

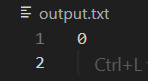
1)





2)





|  | Время выполнения, с | Затраты памяти, МБ |
| --- | --- | --- |
| Пример из задачи | 0.0020027161 | 0.017978 |
| Пример из задачи | 0.0021336079 | 0.017994 |

Вывод по задаче: Программа корректно работает на всех приведенных тестах и укладывается в ограничения по времени и памяти

раничения по времени и памяти

# Вывод

В ходе данной лабораторной работы я научился решать задачи. Написанные программы были протестированы, а также были измерены потребляемый ими объём памяти и время работы. Все программы работаю корректно и укладываются в установленные ограничения по времени и памяти на примерах из задач.