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 5

Выполнил:
Егоров Алексей Алексеевич
К3141

Проверила:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

Задача №2. Карта [1 баллов]

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.
- Пример:

input.txt	output.txt	input.txt	output.txt
treasure	8	you will never find the treasure	146

- Проверяем **обязательно** – на OpenEdu, курс Алгоритмы программирования и структуры данных, неделя 9, задача 2.

Листинг кода

```
import time
import tracemalloc


def treasure(string):
    i = 0

    while i < len(string):
        if string[i] == ' ':
            string = string[:i] + string[i + 1:]
        else:
            i += 1
```

```
    prev_chars = {}
    char_tup = [(0, 0) for _ in range(len(string))]

    for i in range(len(string)):
        if string[i] in prev_chars:
            prev_index = prev_chars[string[i]]
            prev_state = char_tup[prev_index]
              char_tup[i] = (prev_state[0] + 1, prev_state[1] + (i - 1 - prev_index) +
(prev_state[0] * (i - prev_index)))
        prev_chars[string[i]] = i

    result = 0
    for state in char_tup:
        result += state[1]
    return result


def main():
    with open('input.txt') as f:
        string = f.readline().strip()

    with open("output.txt", "w") as f:
        f.write(str(treasure(string)))


if __name__ == "__main__":
    tracemalloc.start()
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

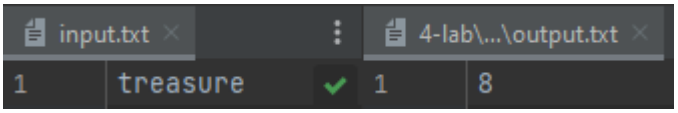
Текстовое объяснение решения

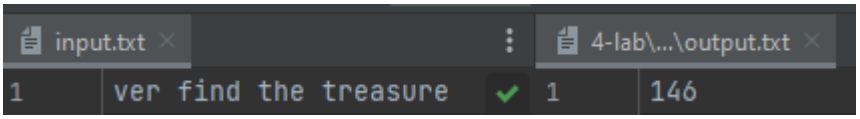
Функция treasure принимает на вход строку string. Если в string есть пробелы, они удаляются. Инициализируются словарь prev_chars для хранения последних найденных индексов конкретных букв, и список кортежей char_spec, который хранит количество вхождений каждой буквы в строку и количество возможных вариантов получения симметричной строки, в которой будут присутствовать две одинаковые эти буквы.

Далее в цикле проверяется, есть ли такая буква в списке prev_chars. Если да, то функция берёт индекс такой же буквы, встретившейся ранее, и кортеж из списка char_spec для этой буквы, а затем рассчитывает новое «состояние»: увеличивает количество этих букв на единицу и пересчитывает количество вариантов, складывая предыдущее количество вариантов, количество вариантов относительно последней и предпоследней буквы и количество вариантов относительно последней буквы с учётом предыдущего количества вариантов. В prev_chars заносится индекс этой буквы.

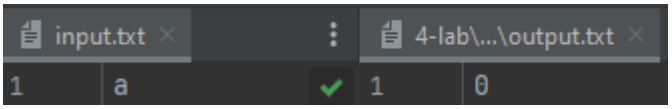
Затем все варианты относительно всех букв складываются, это значение функция возвращает.

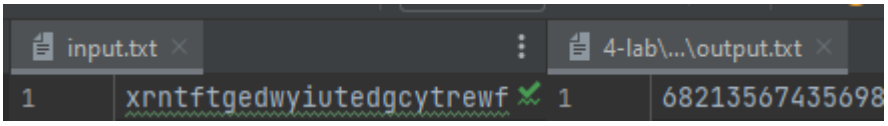
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.0009972000261768699	18,06
Пример из задачи	0.0004279999993741 51	17,99

	23	
Пример из задачи	0.000405499944463372 23	18,17
Верхняя граница диапазона значений входных данных из текста задачи	1.193458345786954873 4	35934,24

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

Задача №3. Паттерн в тексте [1 баллов]

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.
- Примеры:

input	output	input	output	input	output
aba	2	Test	1	aaaaa	3
abacaba	1 5	testTesttesT	5	baaaaaaa	2 3 4

- В первом примере паттерн *aba* можно найти в позициях 1 (**aba**caba) и 5 (abac**aba**) текста *abacaba*.

  Паттерн и текст в этой задаче чувствительны к регистру. Поэтому во втором примере паттерн *Test* встречается только в 45 позиции в тексте *testTesttesT*.

  Обратите внимание, что вхождения шаблона в тексте могут перекрываться, и это нормально, вам все равно нужно вывести их все.
- Используйте оператор == в Python вместо реализации собственной функции AreEqual для строк, потому что встроенный оператор == будет работать намного быстрее.
- Проверяем **обязательно** – на OpenEdu, курс Алгоритмы программирования и структуры данных, неделя 9, наблюдаемая задача.

Листинг кода

```
import time
import tracemalloc
import random
```

```
def poly_hash(S, p, x):
    h = 0
    for i in range(len(S)):
        h += ord(S[i]) * (x ** i)
    return h % p


def precalculate_hashes(T, P, p, x):
    len_t = len(T)
    len_p = len(P)
    H = [0] * (len_t - len_p + 1)
    S = T[len_t - len_p:len_t]
    H[len_t - len_p] = poly_hash(S, p, x)

    y = 1
    for i in range(len_p):
        y = (y * x) % p
    for i in range(len_t - len_p - 1, -1, -1):
        H[i] = (x * H[i + 1] + ord(T[i]) - y * ord(T[i + len_p])) % p
    return H


def alg_rabin_karp(T, P):
    p = 10 ** 9 + 9
    x = random.randint(1, p - 1)
    result = []
    pol_hash = poly_hash(P, p, x)
    H = precalculate_hashes(T, P, p, x)
    for i in range(len(T) - len(P) + 1):
        if pol_hash != H[i]:
            continue
        if T[i:i + len(P)] == P:
            result.append(i + 1)

    return result
```

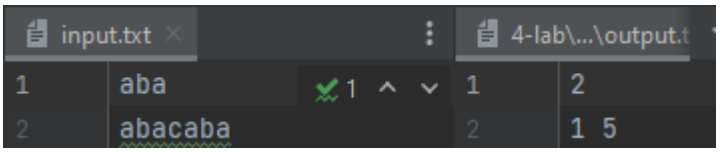
```
def main():
    with open('input.txt') as f:
        P = f.readline().strip()
        T = f.readline().strip()

    indexes = alg_rabin_karp(T, P)
    with open("output.txt", "w") as f:
        f.write(f"{len(indexes)}\n")
        f.write(" ".join(map(str, indexes)))


if __name__ == "__main__":
    tracemalloc.start()
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

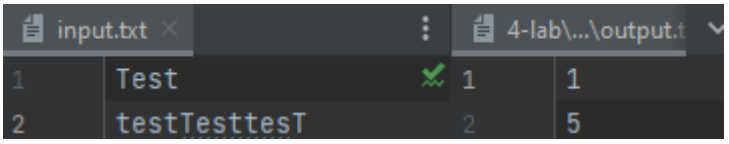
Текстовое объяснение решения

В данной задаче реализован алгоритм Рабина-Карпа, представленный в лекции.

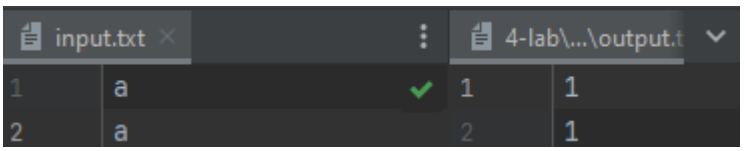
Функция alg_rabin_karp принимает на вход строку T и шаблон P. Присваивается огромное простое число переменной p и выбирается произвольно, но не больше p, число x, которые необходимы для полиномиальной хеш-функции. Далее рассчитывается хеш-значение шаблона, и предварительно хеш-значения для подстрок исходной строки. Затем в цикле сравниваются хеш-значения подстрок и шаблона. Если она равны, то дополнительно (т. к. могла возникнуть коллизия) подстрока и шаблон сравниваются посимвольно, после чего, при положительном результате индекс, с которого начинается текущая подстрока, добавляется в список result. В противном случае происходит переход к следующему символу. Функция возвращает список индексов result.

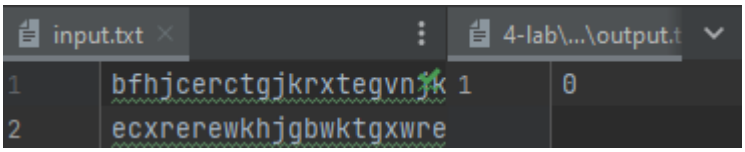
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.000360000636652112	17,72
Пример из задачи	0.004367700079455972	17,84
Пример из задачи	0.0004256999818608165	17,85
Пример из задачи	0.0004332000389695167 5	17,84
Верхняя граница диапазона значений входных данных из текста задачи	1.8923456456456268564	259,43

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

Задача №8. Шаблоны с несовпадениями [2 баллов]

Естественным обобщением задачи сопоставления паттернов, текстов является следующее: найти все места в тексте, расстояние (различие) от которых до образца достаточно мало. Эта проблема находит применение в текстовом поиске (где несовпадения соответствуют опечаткам) и биоинформатике (где несовпадения соответствуют мутациям).

В этой задаче нужно решить следующее. Для целочисленного параметра k и двух строк $t = t_0t_1...t_{m-1}$ и $p = p_0p_1...p_{n-1}$, мы говорим, что p встречается в t в знаке индекса i с не более чем k несовпадениями, если строки p и $t[i : i + p) = t_it_{i+1}...t_{i+n-1}$ различаются не более чем на k знаков.

- **Формат ввода / входного файла (input.txt).** Каждая строка входных данных содержит целое число k и две строки t и p, состоящие из строчных латинских букв.
- **Ограничения на входные данные.** $0 \le k \le 5$, $1 \le |t| \le 200000$, $1 \le |p| \le \min|t|, 100000$. Суммарная длина строчек t не превышает 200 000, общая длина всех p не превышает 100 000.
- **Формат вывода / выходного файла (output.txt).** Для каждой тройки (k, t, p) найдите все позиции $0 \le i_1 < i_2 < ... < i_l < |t|$ в которых строка p встречается в строке t с не более чем k несоответствиями. Выведите l и $i_1, i_2, ..., i_l$.
- Ограничение по времени. 40 сек. (Python), 2 сек (C++).
- Ограничение по памяти. 512 мб.
- Пример:

input	output
0 ababab baaa	0
1 ababab baaa	1 1
1 xabcabc ccc	0
2 xabcabc ccc	4 1 2 3 4
3 aaa xxx	1 0

- Объяснение:

  Для первой тройки точных совпадений нет. Для второй тройки baaa находится на расстоянии один от паттерна с началом в индексе 1 *ababab*. Для третьей тройки нет вхождений не более чем с одним несовпадением. Для четвертой тройки любая (длина три) подстрока p, содержащая хотя бы одну букву c, находится на расстоянии не более двух от t. Для пятой тройки t и p различаются тремя позициями.

- Начните с вычисления хеш-значений префиксов t и p и их частичных сумм. Это позволяет сравнивать любые две подстроки t и p за ожидаемое постоянное время. Для каждой позиции-кандидата i выполните k шагов вида «найти следующее несоответствие». Каждое такое несоответствие можно найти с помощью бинарного поиска.

Листинг кода

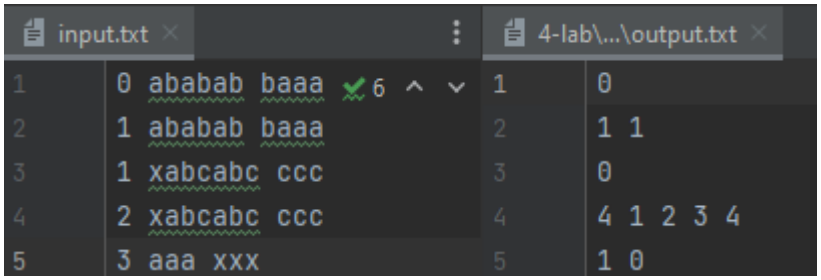
```
import time
import tracemalloc

def main():
    with open("input.txt", "r") as f:
        s = [i.strip() for i in f.readlines()]

    with open("output.txt", "w") as f:
        for o in s:
            count = 0
            res = ""
            k, t, p = o.split()
```

```
            k = int(k)
            for i in range(len(t) - len(p) + 1):
                error = 0
                for h in range(i, i + len(p)):
                    if p[h - i] != t[h]:
                        error += 1
                        if error > k:
                            break
                if error <= k:
                    count += 1
                    res += str(i) + " "
            f.write(f"{count} {res}\n")


if __name__ == "__main__":
    tracemalloc.start()
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

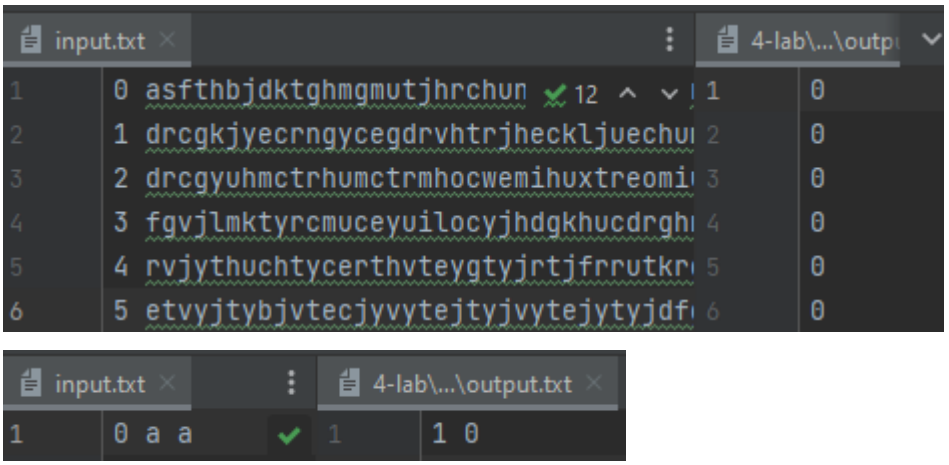
Текстовое объяснение решения

После считывания данных, для каждой строки алгоритм проходит по всем возможным позициям начала подстроки p в строке t. Для каждой позиции он подсчитывает количество несовпадающих символов между текущим фрагментом строки t и шаблоном p. Если количество несовпадающих символов не превышает допустимое значение k, позиция считается подходящей.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:



	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.0004138000076636672	18,81
Пример из задачи	0.0061302999965846545	19,07
Верхняя граница диапазона значений входных данных из текста задачи	3.8234545276568763455	449,34

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

Дополнительные задачи

Задача №1. Наивный поиск подстроки в строке [1 баллов]

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.** $1 \leq |p|, |t| \leq 10^4$.
- **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.
- Пример:

input.txt	output.txt
aba abaCaba	2 1 5

- Проверяем **обязательно** – на OpenEdu, курс Алгоритмы программирования и структуры данных, неделя 9, задача 1.

Листинг кода.

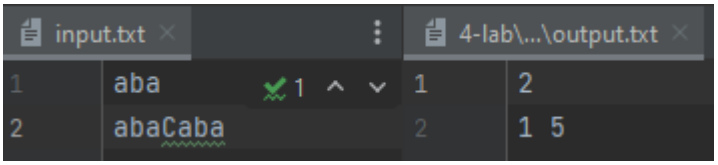
```
import time
import tracemalloc


def find_sub_string(sub_string, main_string):
    len_sub_string = len(sub_string)
    len_main_string = len(main_string)
    count = 0
    positions = []
    for i in range(len_main_string - len_sub_string + 1):
        if main_string[i: i + len_sub_string] == sub_string:
            count += 1
            positions.append(i + 1)
    return count, positions


def main():
    with open('input.txt') as f:
        p = f.readline().strip()
        t = f.readline().strip()
```

```
    count, positions = find_sub_string(p, t)
    with open("output.txt", "w") as f:
        f.write(f"{count}\n")
        f.write(' '.join(map(str, positions)))


if __name__ == "__main__":
    tracemalloc.start()
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

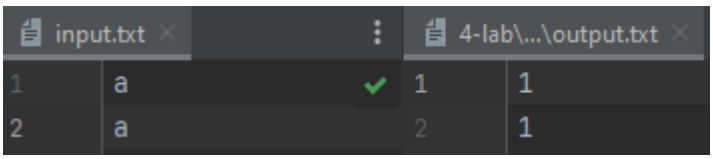
Текстовое объяснение решения.

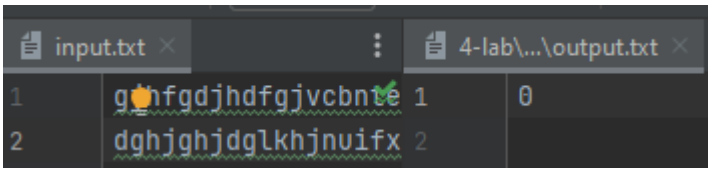
Функция find_sub_string принимает на вход шаблон substr и строку string. Длины шаблона и строки заносятся в соответствующие переменные len_sub_string и len_main_string. Затем в цикле последовательно проверяется, равны ли подстрока длины len_sub_string, которая начинается с индекса i, и шаблон. Если да, то счётчик count увеличивается на единицу, а в список positions добавляется текущий индекс i+1 (т. к. по условию задачи индексы в строке начинаются с 1). Функция возвращает count и список positions.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.0003784000873565674	18,54
Пример из задачи	0.00509380002040416	18,65
Верхняя граница диапазона значений входных данных из текста задачи	0.024005500017665327	50,96

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

Задача №4. Равенство подстрок [1,5 баллов]

В этой задаче вы будете использовать хеширование для разработки алгоритма, способного предварительно обработать заданную строку s, чтобы ответить эффективно на любой запрос типа «равны ли эти две подстроки s?» Это, в свою очередь, является основной частью во многих алгоритмах обработки строк.

- **Формат ввода / входного файла (input.txt).** Первая строка содержит строку s, состоящую из строчных латинских букв. Вторая строка содержит количество запросов q. Каждая из следующих q строк задает запрос тремя целыми числами a, b и l.
- **Ограничения на входные данные.** $1 \le |s| \le 500000, 1 \le q \le 100000, 0 \le a, b \le |s| - l$ (следовательно, индексы a и b начинаются с 0).
- **Формат вывода / выходного файла (output.txt).** Для каждого запроса выведите «Yes», если подстроки $s_a s_{a+1} ... s_{a+l-1} = s_b s_{b+1} ... s_{b+l-1}$ равны, и «No» – если не равны.
- Ограничение по времени. 10 сек.
- Ограничение по памяти. 512 мб.
- Пример:

input	output
trololo 4 0 0 7 2 4 3 3 5 1 1 3 2	Yes Yes Yes No

- Что в примере?
  - 0 0 7 → trololo = trololo
  - 2 4 3 → trololo = trololo
  - 3 5 1 → trololo = trololo
  - 1 3 2 → trololo ≠ trololo
- Что делать?

  Для строки $t = t_0 t_1 ... t_{m-1}$ длиной m и для целого числа x определим полиномиальную хеш-функцию:

$$H(t) = \sum_{j=0}^{m-1} t_j x^{m-j-1} = t_0 x^{m-1} + t_1 x^{m-2} + ... + t_{m-2} x + t_{m-1}.$$

  Пусть $s_a s_{a+1} ... s_{a+l-1}$ – это подстрока заданной строки $s = s_0 s_1 ... s_{n-1}$. Замечательным свойством полиномиальной хеш-функции H является то, что $H(s_a s_{a+1} ... s_{a+l-1})$ можно выразить через $H(s_0 s_1 ... s_{a+l-1})$ и $H(s_0 s_1 ... s_{a-1})$, т. е. через хеш-значения двух префиксов s:

$$\begin{aligned} H(s_a s_{a+1} ... s_{a+l-1}) &= s_a x^{l-1} + s_{a+1} x^{l-2} + ... + s_{a+l-1} = \\ &= s_0 x^{a+l-1} + s_1 x^{a+l-2} + ... + s_{a+l-1} - \\ &- x^l (s_0 x^{a-1} + s_1 x^{a-2} + ... + s_{a-1}) = \\ &= H(s_0 s_1 ... s_{a+l-1}) - x^l H(s_0 s_1 ... s_{a-1}). \end{aligned}$$

  Это приводит нас к следующей идее: мы предварительно вычисляем и сохраняем хэш-значения всех префиксов s: пусть $h[0] = 0$ и, для $1 \le i \le n$, пусть $h[i] = H(s_0 s_1 ... s_{i-1})$. Тогда тождество, приведенное выше, становится

$$H(s_a s_{a+1} ... s_{a+l-1}) = h[a+l] - x^l h[a].$$

Другими словами, мы можем получить хеш-значение любой подстроки s всего за константное время! Ясно, что если $H(s_a s_{a+1} \ldots s_{a+l-1}) \neq H(s_b s_{b+1} \ldots s_{b+l-1})$, то соответствующие две подстроки ($s_a s_{a+1} \ldots s_{a+l-1}$ и $s_b s_{b+1} \ldots s_{b+l-1}$) различны. Однако, если хеш-значения совпадают, все же возможно, что подстроки различаются – это называется *коллизией*. Ниже мы обсудим, как уменьшить вероятность коллизии.

Напомним, что на практике никогда не вычисляется точное значение полиномиальной хеш-функции: все вычисляется по модулю m для некоторого фиксированного целого числа m. Это делается для того, чтобы все вычисления были эффективными, а хэш-значения были достаточно малы. Напомним также, что при вычислении $H(s) \mod m$ важно делать каждый промежуточный шаг (а не окончательный результат) по модулю m.

Можно показать, что если s_1 и s_2 – две *разные* строки длины n, а m – простое целое число, то вероятность того, что $H(s_1) \mod m = H(s_2) \mod m$ (при выборе $0 \leq x \leq m-1$) не более чем $\frac{n}{m}$ (примерно, это потому, что $H(s_1) - H(s_2)$ является ненулевым полиномом степени не выше $n-1$ и, следовательно, может иметь не более n корней по модулю m). Чтобы еще больше снизить вероятность столкновения, можно взять два разных модуля.

В целом получается следующий подход.

1. Установите $m_1 = 10^9 + 7$ и $m_2 = 10^9 + 9$.
2. Выберите случайный x от 1 до 10^9.
3. Посчитайте хеш-таблицы $h_1[0..n]$ и $h_2[0..n]$: $h_1[0] = h_2[0] = 0$ и для всех $1 \leq i \leq n$: $h_1[i] = H(s_0 \ldots s_{i-1}) \mod m_1$ и $h_2[i] = H(s_0 \ldots s_{i-1}) \mod m_2$. Проиллюстрируем это для h_1. Используя $h_1[0] = 0$ в цикле от 1 до n будем считать:
$$h_1[i] \leftarrow (x \cdot h_1[i-1] + s_i) \mod m_1$$
4. Для каждого запроса (a, b, l):
   (a) Используйте предварительно вычисленные хеш-значения, чтобы вычислить хеш-значения подстрок $s_a s_{a+1} \ldots s_{a+l-1}$ и $s_b s_{b+1} \ldots s_{b+l-1}$ по модулю m_1 и m_2.
   (b) Выведите «Yes», если
$$H(s_a s_{a+1} \ldots s_{a+l-1}) \mod m_1 = H(s_b s_{b+1} \ldots s_{b+l-1}) \mod m_1, \text{ и}$$
$$H(s_a s_{a+1} \ldots s_{a+l-1}) \mod m_2 = H(s_b s_{b+1} \ldots s_{b+l-1}) \mod m_2$$
   (c) В противном случае выведите «No»

Обратите внимание, что, в отличие от алгоритма Карпа–Рабина, мы не сравниваем подстроки наивно, когда их хеши совпадают. Вероятность этого события не превышает $\frac{n}{m_1} \cdot \frac{n}{m_2} \leq 10^{-9}$. (На самом деле для случайных строк вероятность даже намного меньше: 10^{-18}.)

Листинг кода.

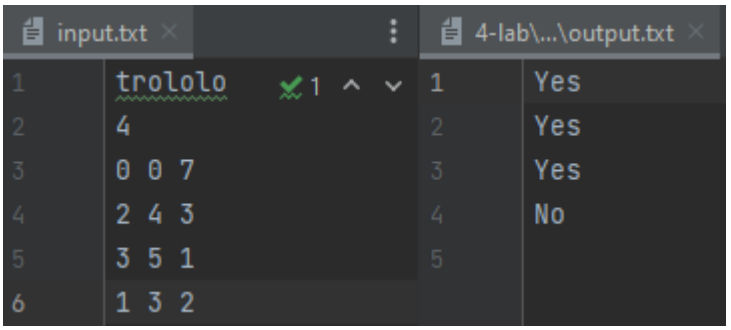
```
import time
import tracemalloc
import random


def precalculate_hashes(s, x, m):
    n = len(s)
    H = [0] * n
    for i in range(1, n):
        H[i] = (x * H[i-1] + ord(s[i])) % m
    return H


def are_equal(s, a, b, l, x, H1, H2, m1, m2):
    p = x**l
    h1_a = (H1[a+l-1] - p * H1[a-1]) % m1
    h2_a = (H2[a+l-1] - p * H2[a-1]) % m2
```

```
    h1_b = (H1[b+l-1] - p * H1[b-1]) % m1
    h2_b = (H2[b+l-1] - p * H2[b-1]) % m2
    if h1_a == h1_b and h2_a == h2_b:
        return "Yes"
    return "No"


def main():
    with open('input.txt') as f_input, open("output.txt", "w") as f_output:
        s = f_input.readline().strip()
        q = int(f_input.readline())
        m1 = 10 ** 9 + 7
        m2 = 10 ** 9 + 9
        x = random.randint(1, 10 ** 9)
        H1 = precalculate_hashes(s, x, m1)
        H2 = precalculate_hashes(s, x, m2)
        for _ in range(q):
            a, b, l = map(int, f_input.readline().split())
            f_output.write(are_equal(s, a, b, l, x, H1, H2, m1, m2) + "\n")


if __name__ == "__main__":
    tracemalloc.start()
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

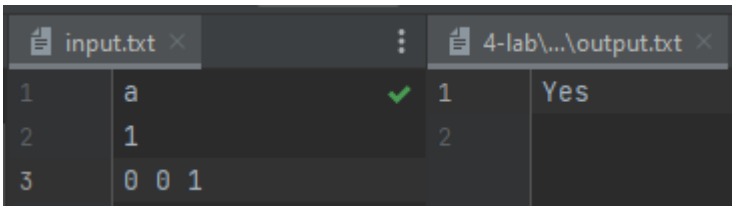
Текстовое объяснение решения.

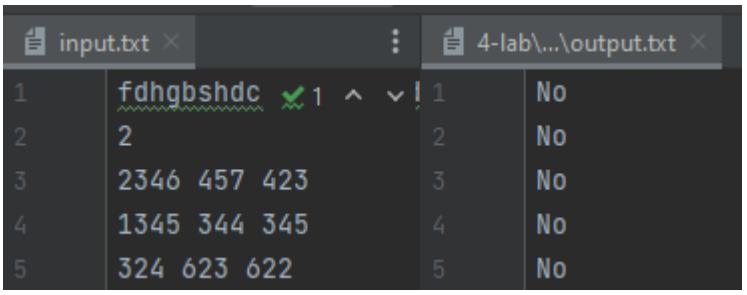
Предварительно с помощью полиномиальной хеш-функции рассчитываются две хеш-таблицы для входящей строки с разными значениями делителя по модулю m1 и m2 для хеш-функции. Затем для каждого запроса (a, b, l) вызывается функция are_equal. Она по формуле из текста задачи рассчитывает по два хеш-значения для двух подстрок. Если оба хеш-значения одной подстроки будут соответственно равны двум

хешзначениям другой подстроки, то это будет означать, что подстроки одинаковые, и функция вернёт значение «Yes», иначе – «No».
Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.0003961001057177782	28,49
Пример из задачи	0.0010696001118049026	30,10
Верхняя граница диапазона значений входных данных из текста задачи	11,212465456324957678	34984,63

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

Задача №5. Префикс-функция [1,5 баллов]

Постройте префикс-функцию для всех непустых префиксов заданной строки s.

- **Формат ввода / входного файла (input.txt).** Одна строка входного файла содержит s. Строка состоит из букв латинского алфавита.
- **Ограничения на входные данные.** $1 \le |s| \le 10^6$.
- **Формат вывода / выходного файла (output.txt).** Выведите значения префикс-функции для всех префиксов строки s длиной $1, 2, ..., |s|$, в указанном порядке.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Примеры:

input.txt	output.txt	input.txt	output.txt
aaaAAA	0 1 2 0 0 0	abacaba	0 0 1 0 1 2 3

- Проверяем **обязательно** – на OpenEdu, курс Алгоритмы программирования и структуры данных, неделя 10, задача 1.

Листинг кода.

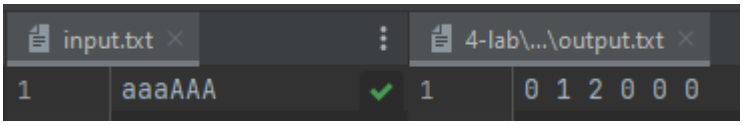
```
import time
import tracemalloc


def prefix_function(s):
    P = [0] * (len(s) + 1)
    i, j = 1, 0
    while i < len(s):
        if s[i] == s[j]:
            P[i+1] = j + 1
            i += 1
            j += 1
        else:
            if j <= 0:
                P[i + 1] = 0
                i += 1
            else:
                j = P[j]
    return P[1:]
```

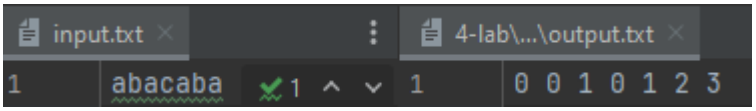
```
def main():
    with open('input.txt') as f:
        s = f.readline()

    with open("output.txt", "w") as f:
        f.write(' '.join(map(str, prefix_function(s))))


if __name__ == "__main__":
    tracemalloc.start()
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

Текстовое объяснение решения.

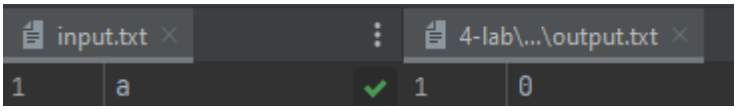
Функция prefix_function принимает на вход строку s. Инициализируется список P, в котором хранятся длины наибольших бордеров для каждой позиции строки (длина наибольшего префикса строки, являющегося также её суффиксом). Создаются две переменные i и j для хранения индексов. Далее в цикле пока не пройдена вся строка происходит проверка двух символов строки. Если они равны, то в список P для текущего символа строки записывается значение j + 1 (длина бордера увеличивается), значения j и i увеличиваются на единицу. Если символы не равны, то если j больше 0, то в j записывается значение элемента с индексом j из списка P («сдвигаем шаблон»), иначе элемент списка P с индексом i+1 записывается 0, и индекс i увеличивается на единицу. Функция возвращает список P без первого элемента

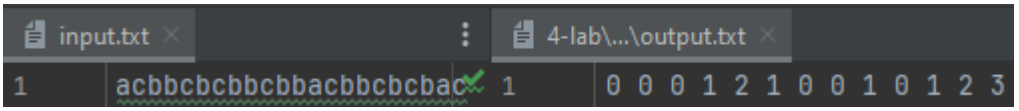
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.0003282000543549657	18,06
Пример из задачи	0.00434410001616925	18,12
Пример из задачи	0.0003308999584987759 6	18,12
Верхняя граница диапазона значений входных данных из текста задачи	1.8934532949853497634	64953,47

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

Задача №6. Z-функция [1.5 баллов]

Постройте Z-функцию для заданной строки s.

- **Формат ввода / входного файла (input.txt).** Одна строка входного файла содержит s. Строка состоит из букв латинского алфавита.
- **Ограничения на входные данные.** $2 \le |s| \le 10^6$.
- **Формат вывода / выходного файла (output.txt).** Выведите значения Z-функции для всех индексов $1, 2, ..., |s|$ строки s, в указанном порядке.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Примеры:

input.txt	output.txt	input.txt	output.txt
aaaAAA	2 1 0 0 0	abacaba	0 1 0 3 0 1

- Проверяем **обязательно** – на OpenEdu, курс Алгоритмы программирования и структуры данных, неделя 10, задача 2.

Листинг кода.

```
import time
import tracemalloc


def z_function(s):
    z_f = [0] * len(s)
    left = right = 0
    for i in range(1, len(s)):
        z_f[i] = max(0, min(right - i, z_f[i - left]))
        while i + z_f[i] < len(s) and s[z_f[i]] == s[i + z_f[i]]:
            z_f[i] += 1
        if i + z_f[i] > right:
            left = i
            right = i + z_f[i]
    return z_f[1:]


def main():
    with open('input.txt') as f:
        s = f.readline()

    with open("output.txt", "w") as f:
        f.write(' '.join(map(str, z_function(s))))
```

```
if __name__ == "__main__":
    tracemalloc.start()
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

Текстовое объяснение решения.

Функция z_function принимает на вход строку s. Инициализируется список z_f, в котором хранятся z-значения. Создаются две переменные left и right для хранения индексов.

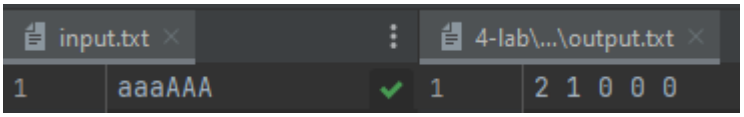
Далее выполняется итерация по строке, начиная с индекса 1 (поскольку значение z в индексе 0 равно 0). Для каждого индекса i вычисляется значение z на основе ранее вычисленных значений и текущей позиции.

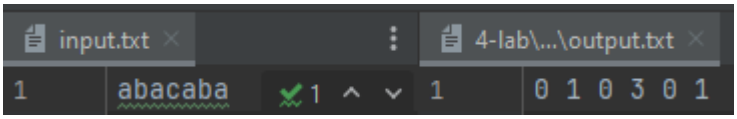
Затем берётся минимум среди значений right-i и z_f[i-left]. Это гарантирует, что мы не выйдем за границу рассматриваемого отрезка совпадения и не будем повторно рассматривать символы. Если минимум отрицательный, то берётся 0.

Потом в цикле, пока отрезок совпадения не дошёл до конца строки, и символы в текущей позиции плюс значение z и в соответствующей позиции в префиксе не перестали совпадать, увеличиваем z-значение. Далее если мы перешли правую границу отрезка рассмотрения, то левая и правая границы смещаются.

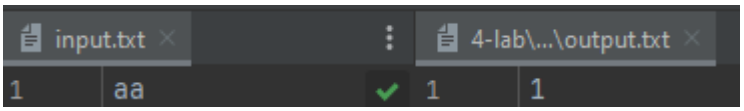
Функция возвращает z-значения, начиная с индекса 1, исключая значение с индексом 0, поскольку оно было инициализировано равным 0 и не представляет значимой информации о структуре строки.

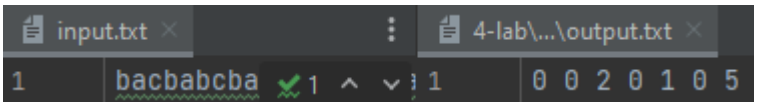
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.0003619999624788761	18,11
Пример из задачи	0.0003742000553756952 3	18,12
Пример из задачи	0.0004968000575900078	18,12
Верхняя граница диапазона значений входных данных из текста задачи	3.234134546234567634 3	64534,43

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

Задача №7. Наибольшая общая подстрока [2 баллов]

В задаче на наибольшую общую подстроку даются две строки s и t, и цель состоит в том, чтобы найти строку w максимальной длины, которая является подстрокой как s, так и t. Это естественная мера сходства между двумя строками. Задача имеет применения для сравнения и сжатия текстов, а также в биоинформатике. Эту проблему можно рассматривать как частный случай проблемы расстояния редактирования (Левенштейна), где разрешены только вставки и удаления. Следовательно, ее можно решить за время $O(|s||t|)$ с помощью динамического программирования. Есть также весьма нетривиальные структуры данных для решения этой задачи за линейное время $O(|s| + |t|)$. В этой задаче ваша цель – использовать хеширование для решения почти за линейное время.

- **Формат ввода / входного файла (input.txt).** Каждая строка входных данных содержит две строки s и t, состоящие из строчных латинских букв.
- **Ограничения на входные данные.** Суммарная длина всех s, а также суммарная длина всех s не превышает 100 000.
- **Формат вывода / выходного файла (output.txt).** Для каждой пары строк s_i и t_i найдите ее самую длинную общую подстроку и уточните ее параметры, выведя три целых числа: ее начальную позицию в s, ее начальную позицию в t (обе считаются с 0) и ее длину. Формально выведите целые числа $0 \le i < |s|$, $0 \le j < |t|$ и $l \ge 0$ такие, что и l максимально. (Как обычно, если таких троек с максимальным l много, выведите любую из них.)
- Ограничение по времени. 15 сек.
- Ограничение по памяти. 512 мб.
- Пример:

input	output
cool toolbox	1 1 3
aaa bb	0 1 0
aabaa babbaab	0 4 3

- Объяснение:

  Самая длинная общая подстрока первой пары строк – *ool*, она начинается с первой позиции в *toolbox* и с первой позиции в *cool*. Строки из второй строки не имеют общих непустых общих подстрок (в этом случае $l = 0$ и можно вывести любые индексы i и j). Наконец, последние две строки имеют общую подстроку *aab* длины 3, начинающуюся с позиции 0 в первой строке и с позиции 4 во второй. Обратите внимание, что для этой пары строк также можно вывести 2 3 3.

- Что делать?

  Для каждой пары строк s и t используйте двоичный поиск, чтобы найти длину наибольшей общей подстроки. Чтобы проверить, есть ли у двух строк общая подстрока длины k,

  - предварительно вычислить хеш-значения всех подстрок длины k из s и t;
  - обязательно используйте несколько хэш-функций (но не одну), чтобы уменьшить вероятность коллизии;
  - храните хеш-значения всех подстрок длины k строки s в хеш-таблице; затем пройдитесь по всем подстрокам длины k строки t и проверьте, присутствует ли хеш-значение этой подстроки в хеш-таблице.

Листинг кода.

```
import time
import tracemalloc
import random


def poly_hash_1(S, p, x):
    h = 0
    for i in range(len(S)):
        h += ord(S[i]) * (x**i)
    return h % p
```

```
def poly_hash_2(S, p, x):
    h = 0
    for i in range(len(S)):
        h += (ord(S[i]) - 65) * (x**(i+1))
    return h % p


def exist(s, t, i, p, x):
    if i == 0:
        return (True, 0, 0)

    hashes_set_1 = set()
    hashes_set_2 = set()
    hashes = [0] * (len(s) - i + 1)
    for j in range(len(s) - i + 1):
        h1 = poly_hash_1(s[j:j + i], p, x)
        h2 = poly_hash_2(s[j:j + i], p, x)
        hashes_set_1.add(h1)
        hashes_set_2.add(h2)
        hashes[j] = (j, h1)

    for j in range(len(t) - i + 1):
        hash1 = poly_hash_1(t[j:j+i], p, x)
        hash2 = poly_hash_2(t[j:j+i], p, x)
        if hash1 in hashes_set_1 and hash2 in hashes_set_2:
            for h in hashes:
                if h[1] == hash1:
                    if not (t[j:j+i] == s[h[0]:h[0]+i]):
                        continue
                    return (True, h[0], j)

    return (False, 0, 0)


def biggest_common_substr(s, t):
```

```
    p = 10 ** 9 + 9
    x = random.randint(1, p - 1)
    answer = 0
    low, high = 0, min(len(s), len(t))
    while low <= high:
        mid = (low + high) // 2
        ex = exist(s, t, mid, p, x)
        if ex[0]:
            answer = (ex[1], ex[2], mid)
            low = mid + 1
        else:
            high = mid - 1
    return answer


def main():
    with open('input.txt') as f_input, open("output.txt", "w") as f_output:
        for string in f_input.readlines():
            s, t = string.split()
            f_output.write(' '.join(map(str, biggest_common_substr(s, t))) + '\n')



if __name__ == "__main__":
    tracemalloc.start()
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

Текстовое объяснение решения.

Функция biggest_common_substr принимает на вход две строки s и t. Присваивается огромное простое число переменной p и выбирается произвольно, но не больше p, число x, которые необходимы для

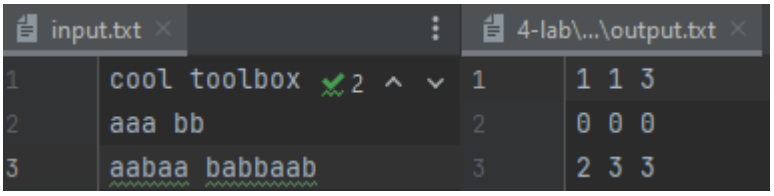
полиномиальной хеш-функции. Далее применяется алгоритм двоичного поиска для нахождения наибольшей длины подстроки. Пока нижняя граница меньше или равна верхней рассчитывается среднее значение mid и вызывается функция exist.

Функция exist принимает на вход две строки s и t, значение длины i, а также p и x. Если длина i равна 0, то функция возвращает кортеж (True, 0, 0), что означает, что подстроки 0 длины всегда равны. В противном случае инициализируются два множества hashes_set_1 и hashes_set_2, в которых хранятся хеш-значения подстрок длины i строки s от двух полиномиальных хеш-функций, а также список hashes, в котором помимо хеш-значений хранятся также индексы, с которых начинаются подстроки.

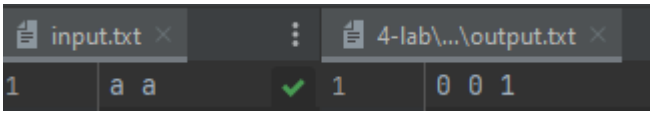
Далее выполняется итерация по строке t. Рассчитываются хешзначения подстрок длины i строки t. Затем проверяется, есть ли такое хешзначение в множествах hashes_set_1 и hashes_set_2. Если да, это означает, что потенциально две строки могут содержать одинаковые подстроки длины i. В списке hashes находится хеш-значение, и затем две подстроки посимвольно сравниваются. При положительном результате функция возвращает кортеж со значениями: True, индекс в строке s и индекс в строке t.

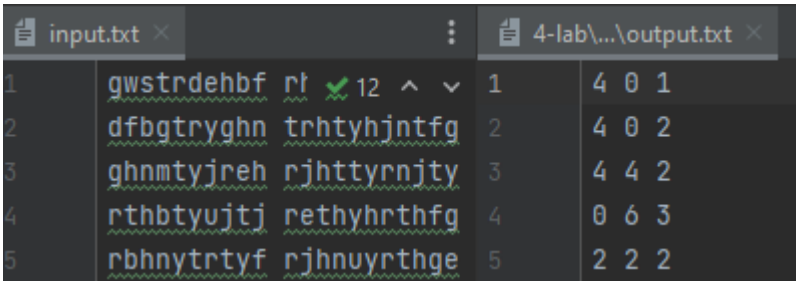
Если функция exist возвращает положительный результат, то индексы подстрок в двух строках и значение длины этих подстрок записываются в answer. Функция biggest_common_substr возвращает answer.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.0027895749857398475	26,58
Пример из задачи	0.0029346973645897342	26,94
Верхняя граница диапазона значений входных данных из текста задачи	2.0234987362457676344	536,43

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

Задача №9. Декомпозиция строки [2 баллов]

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.
- Примеры:

input.txt	output.txt	input.txt	output.txt
ABCABCDEDEDEF	ABC*2+DE*3+F	Hello	Hello

- Проверяем **обязательно** – на OpenEdu, курс Алгоритмы программирования и структуры данных, неделя 10, задача 3.

Листинг кода.

```
import time
import tracemalloc
```

```
def find_decompose(input_str):
    substring = input_str
    builder = []
    start = 0
    while start < len(input_str):
        count = 1
        end = start + 1
        while end < len(input_str):
            substring = input_str[start:end]
            while True:
                next_substring = input_str[start + len(substring):min(end +
                len(substring), len(input_str))]
                if next_substring == substring:
                    count += 1
                    start += len(substring)
                    end += len(substring)
                else:
                    end += 1
                    break
            if count > 1:
                start += len(substring)
                break
        if count > 1:
            if len(substring) * count < len(f'{substring}*{count}+'):
                builder.append(substring * count)
            else:
                if len(builder) > 0 and builder[-1][-1] != '+':
                    builder.append('+')
                builder.append(f"{substring}*{count}+")
        else:
            builder.append(input_str[start])
            start += 1

    result = ''.join(builder)
    if result.endswith('+'):
```

```
        return result[:-1]
    else:
        return result


def main():
    with open('input.txt') as f:
        s = f.readline().strip()

    with open("output.txt", "w") as f:
        f.write(find_decompose(s))


if __name__ == "__main__":
    tracemalloc.start()
    t_start = time.perf_counter()
    main()
    t_end = time.perf_counter()
    print(f"Время: {t_end - t_start} секунд")
    print(f"Память: {tracemalloc.get_traced_memory()}")
    tracemalloc.stop()
```

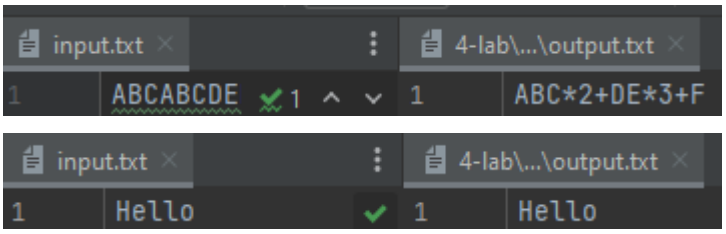
Текстовое объяснение решения.

Функция find_decompose принимает на вход строку input_str. Затем выполняется итерация по input_str со start=0. Устанавливается значение end=start+1. Внутри вложенного цикла берётся подстрока от start до end. Далее проверяется, сколько раз повторяется эта подстрока далее по строке. Если подстрока не повторяется, то граница end сдвигается на единицу вправо, и уже новая подстрока проверяется на повторение.

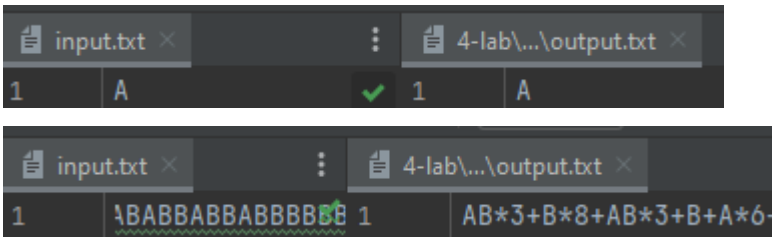
Перед добавлением повторяющейся подстроки в список builder проверяется, не приведёт ли новая запись с умножением подстроки на её количество и добавление знака «+» к более длинной строке, чем простое повторение подстроки. В зависимости от этого в список builder добавляется новая подстрока или прежняя. Если подстрока не повторялась, то символ строки просто добавляется в builder.

Функция возвращает новую сокращённую строку.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:



	Время выполнения, с	Затраты памяти, КБ
Нижняя граница диапазона значений входных данных из текста задачи	0.00029669993091374636	17,99
Пример из задачи	0.004329800023697317	27,75
Пример из задачи	0.0003692000173032284	17,99
Верхняя граница диапазона значений входных данных из текста задачи	3.634534527456453654 5	97,54

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