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 21

Выполнила:
Савченко А.С.
К3141

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

Задача №3. Паттерн в тексте [1 баллов]

Текст задачи:

3 Задача. 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.
- Примеры:

input	output
aba abacaba	2 1 5

input	output
Test testTesttesT	1 5

input	output
aaaaa baaaaaaa	3 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.

```
def rabin_karp(p: str, t: str) -> list[int]:
    len_p, len_t = len(p), len(t)
    if len_p > len_t:
        return []

    result = []
    p_hash, t_hash = hash(p), hash(t[:len_p])

    for i in range(len_t - len_p + 1):
```

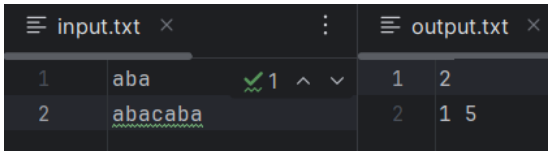
```
        if p_hash == t_hash:
            if p == t[i:i + len_p]:
                result.append(i + 1)
        if i < len_t - len_p:
            t_hash = hash(t[i + 1:i + len_p + 1])
    return result


with open("input.txt", "r", encoding='utf-8') as input_file, open("output.txt", "w", encoding='utf-8') as output_file:
        p, t = input_file.readline().strip(), input_file.readline().strip()
    res = rabin_karp(p, t)
    output_file.write(f'{len(res)}\n{" ".join(map(str, res))}')
```

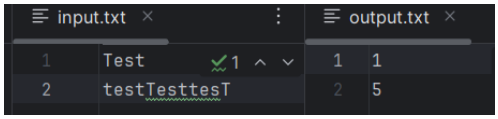
Текстовое объяснение решения.

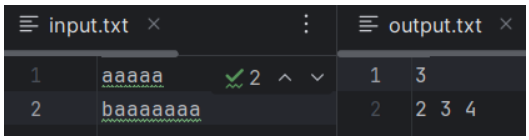
Тут исп. Алгоритм Рабина-Карпа.Хешируем паттерн, хешируем подстроки: последовательно вычисляем хэши всех подстрок в основной строке(дина подстрок = длине паттерна).

Если хэши подстроки и паттерна совпадают, то проверяем, совпадают ли сами строки. Если да - добавляем индекс в результирующий список.

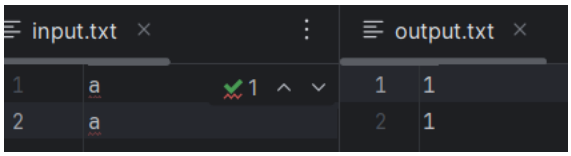
Результат работы кода на примерах из текста задачи:(скрины input output файлов)

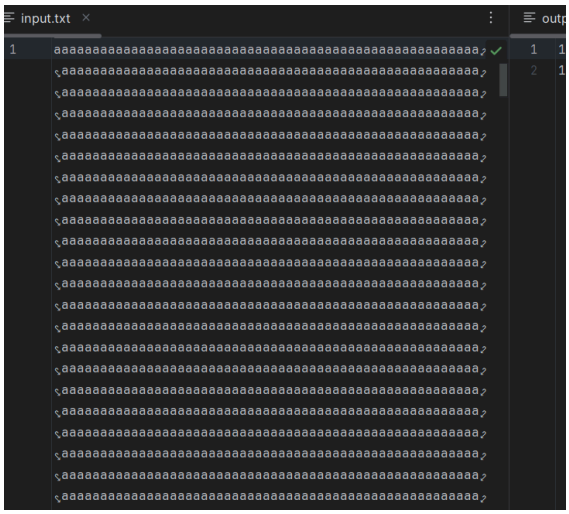






Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)





	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.003547	14.87890625
Пример из задачи	0.0004725	14.75390625
Пример из задачи	0.0033099	14.7578125
Пример из задачи	0.0005252	14.82421875
Верхняя граница диапазона значений входных данных из текста задачи	0.0064105	15.890625

Вывод по задаче:

Задача на применение алгоритма Рабина-Карпа.

Задача №4. Название задачи [1.5 баллов]

Текст задачи:

4 Задача. Равенство подстрок [10 s, 512 Mb, 1.5 балла]

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 $s_a s_{a+1} \ldots s_{a+l-1} = s_b s_{b+1} \ldots s_{b+l-1}$ 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

Листинг кода

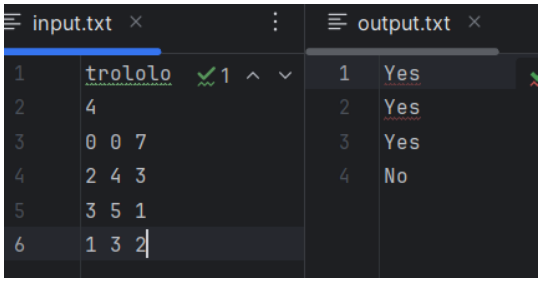
```
def polynom_hash(s, x, m):
    h = [0] * (len(s) + 1)
    for i in range(len(s)):
        h[i + 1] = (h[i] * x + ord(s[i])) % m
    return h


with open("input.txt", "r", encoding='utf-8') as input_file, open("output.txt", "w", encoding='utf-8') as output_file:
    m1, m2 = 10 ** 9 + 7, 10 ** 9 + 9
    x = 1
```

```
    s = input_file.readline().strip()
    h1 = polynom_hash(s, x, m1)
    h2 = polynom_hash(s, x, m2)
    ans = []
    n = len(s)
    q = int(input_file.readline())
    for _ in range(q):
                            a,    b,    l    =    map(int,
input_file.readline().split())
        hash1_a = (h1[a + l] - h1[a] * pow(x, l, m1)) %
m1
        hash1_b = (h1[b + l] - h1[b] * pow(x, l, m1)) %
m1
        hash2_a = (h2[a + l] - h2[a] * pow(x, l, m2)) %
m2
        hash2_b = (h2[b + l] - h2[b] * pow(x, l, m2)) %
m2

        if hash1_a == hash1_b and hash2_a == hash2_b:
            ans.append('Yes')
        else:
            ans.append('No')

    output_file.write('\n'.join(ans))
```

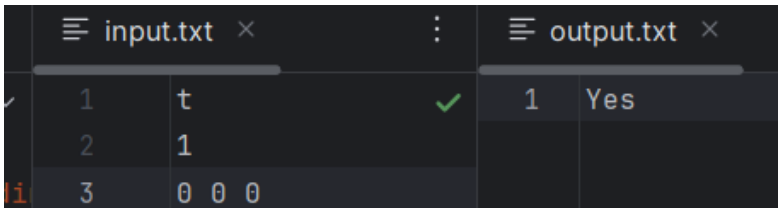
Текстовое объяснение решения:

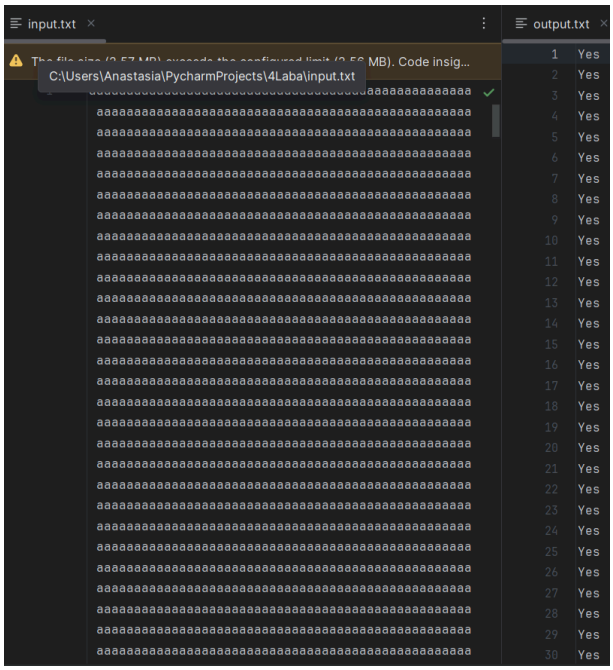
Задаем два простых больших числа m1 m2, с их помощью ,будем вычислять хэш значения подстрок минимизировав вероятность коллизий. Для каждой введенной строки a, b l, алгоритм вычисляет хэши подстрок по двум модулям. Если они совпадают выведите yes иначе no.

Результат работы кода на примерах из текста задачи:(скрины input output файлов)



Результат работы кода на максимальных и минимальных значениях:(скрины input output файлов)





	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0005022	14.88671875
Пример из задачи	0.0005012	14.796875

Верхняя граница диапазона значений входных данных из текста задачи	0.434468	17.1171875

Вывод по задаче:

В этой задаче я реализовать алгоритм полиномиального хэширования. Очень понравился сам алгоритм(подход) решения приведенный в подсказке. То, что мы обрабатываем хэш-таблицы для каждой позиции, позволяет быстро вычислять хэш-значение любой подстроки.

Задача №8. Шаблоны с несовпадениями [2 баллов]

Текст задачи:

8 Задача. Шаблоны с несовпадениями [40 s, 512 Mb, 2 балла]

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:

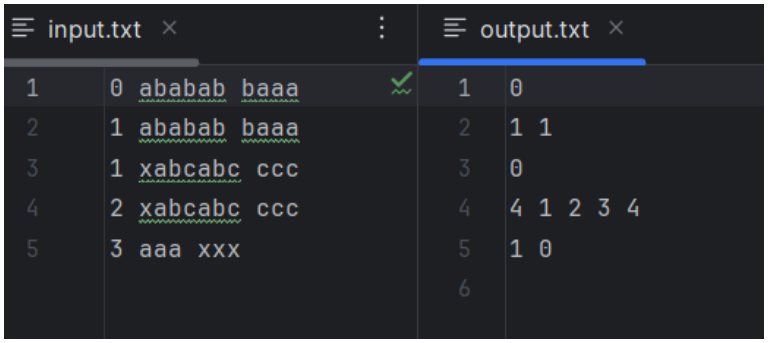
```
def k_mismatch_search(p, t, k):
    matches = []

    for i in range(len(t) - len(p) + 1):
        mismatches = 0
        for j in range(len(p)):
            if t[i + j] != p[j]:
                mismatches += 1
                if mismatches > k:
                    break
```

```
        if mismatches <= k:
            matches.append(i)

    return matches

with open("input.txt", "r", encoding='utf-8') as
input_file, open("output.txt", "w", encoding='utf-8')
as output_file:
    for line in input_file:
        k, t, p = line.strip().split()
        result = k_mismatch_search(p, t, int(k))
                output_file.write(f'{len(result)} {"
".join(map(str, result))}\n')
```

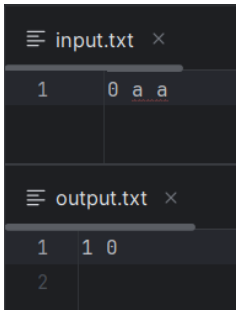
Текстовое объяснение решения:

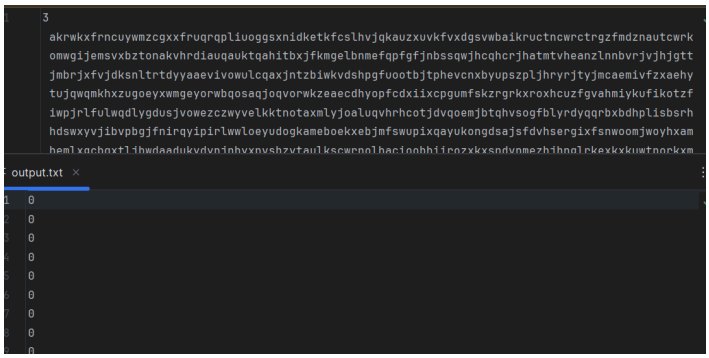
Проходимся по всем позициям главной строки с которых может начинаться подстрока. Инициализируем счетчик несовпадений = 0. Перебираем все символы в паттерне. Сравниваем символы строки и паттерна, если они не совпадают увеличиваем счетчик несовпадений. Если этот счетчик превышает заданное k прерываем цикл, если не превышает добавляем позицию в результирующий список позиций.

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	0.0005331	14.94140625
Пример из задачи	0.0005431	14.83203125
Верхняя граница диапазона значений входных данных из текста задачи	0.3673545	15.29296875

Вывод по задаче:

Изначально я написала алгорит с полиномиальным хэширование, но ,к сожалений, что-то пошло не так и при больших входных данных алгоритм долго работал на этапе поиска совпадений. Так что я остановилась на наивном варианте этого алгорритма без хэширования. По крайней мере он удачно справляется с любыми входными данными.

Дополнительные задачи

Задача №2. Карта [1 баллов]

Текст задачи:

2 Задача. 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.
- Пример:

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

Листинг кода:

```
with open("input.txt", "r", encoding='utf-8') as
input_file, open("output.txt", "w", encoding='utf-8')
as output_file:
    s = ''.join(input_file.read().split())
    prev_char = {}
    my_l = [[0, 0] for _ in range(len(s))]

    for ind, char in enumerate(s):
        prev_i = prev_char.get(char, None)
```

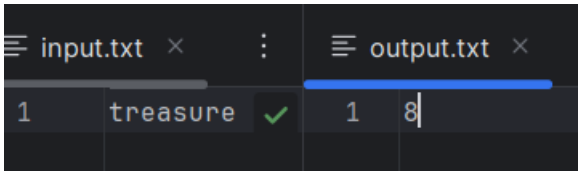
```
        if prev_i is not None:
            my_l[ind][0] = my_l[prev_i][0] + 1
                    my_l[ind][1]  =  my_l[prev_i][1]  +
my_l[prev_i][0] * (ind - prev_i) + (ind - prev_i - 1)
        else:
            my_l[ind] = [0, 0]
        prev_char[char] = ind

    total = sum(i[1] for i in my_l)
    output_file.write(str(total))
```

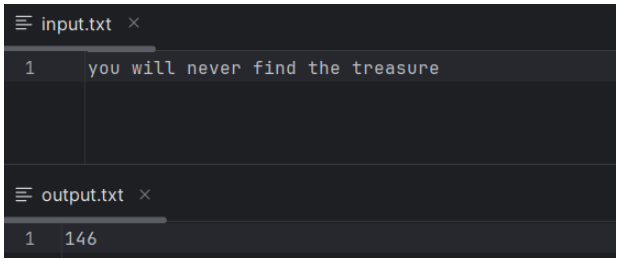
Текстовое объяснение решения:

Так как простой тройной цикл будет работать слишком долго, можно воспользоваться или сторонними библиотеками (например collections.Counter() для подсчета частоты символов или defaultdict), или использовать хэш-таблицы для хранения индексов символов, что я и выбрала.

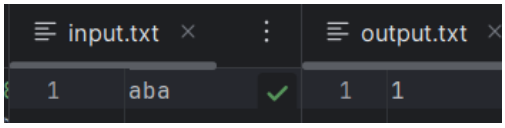
При чтении данных из файла сразу удалим пробелы. Создадим хэш-таблицу(словарь) prev_char, его будем исп-ть для хранения последних индексов символов. Ключ- символы строки, значение — их последний встреченный индекс. Перебираем каждый символ char и его индекс ind в строке s . В цикле для каждого символа, если он уже в словаре - получаем его индекс.После обработки текущего символа обновляем хэш-таблицу, записывая текущий индекс для символа char. Обновляем таблицу my_l[ind][0] чтобы учесть новые палиндромные тройки заканчивающиеся на новом индексе.Если prev_i =None, символ char встречается впервые. В этом случае инициализируем my_l[ind] как [0, 0].

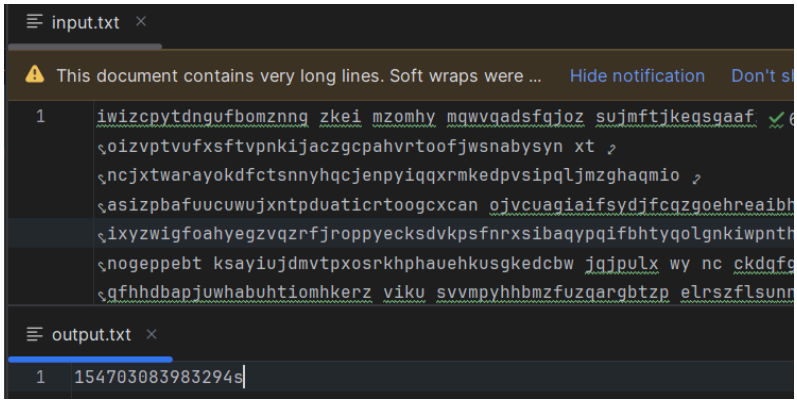
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:





	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0009445	14.875
Пример из задачи	0.0036854	14.890625
Пример из задачи	0.0029547	14.90234375
Верхняя граница диапазона значений входных данных из текста задачи	0.1565675	17.31640625

Вывод по задаче:

Задача довольно интересная. Сначала попробовала простой перебор троек, но на максимальных входных данных он, конечно, не справился.

Задача №5. Префикс-функцияи [1.5 баллов]

Текст задачи:

5 Задача. Префикс-функция [2 s, 256 Mb, 1.5 балла]

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

Листинг кода:

```
def prefix_function(s: str) -> list[int]:
    j = 0
    prefix = [0] * len(s)
    for i in range(1, len(s)):
        j = prefix[i - 1]
        while j > 0 and s[i] != s[j]:
            j = prefix[j - 1]
        if s[i] == s[j]:
            j += 1
        prefix[i] = j
    return prefix
```

```
with open("input.txt", "r", encoding='utf-8') as
input_file, open("output.txt", "w", encoding='utf-8')
as output_file:
    s = input_file.readline().strip()
                output_file.write('     '.join(map(str,
prefix_function(s))))
```

Текстовое объяснение решения:

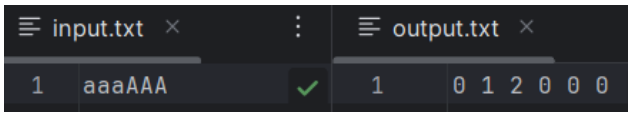
Создаем список значений префикс-функции изначально все элементы равны 0.

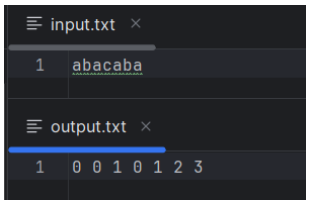
Проходим по строке по индексам i, начиная с первого.

prefix[i] хранит значение префикс-функции для подстроки s[0:i+1]. индексы: i для текущего символа и j для отслеживания текущей длины совпадения.

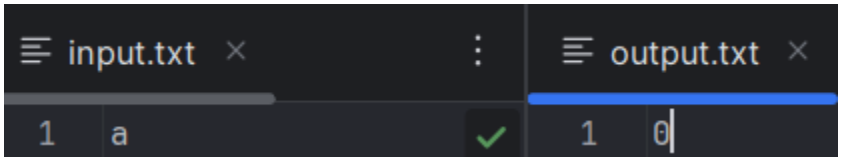
Если символы совпадают, обновляем j и pi[i] иначе уменьшаем j, пока не найдём совпадение или пока j != 0

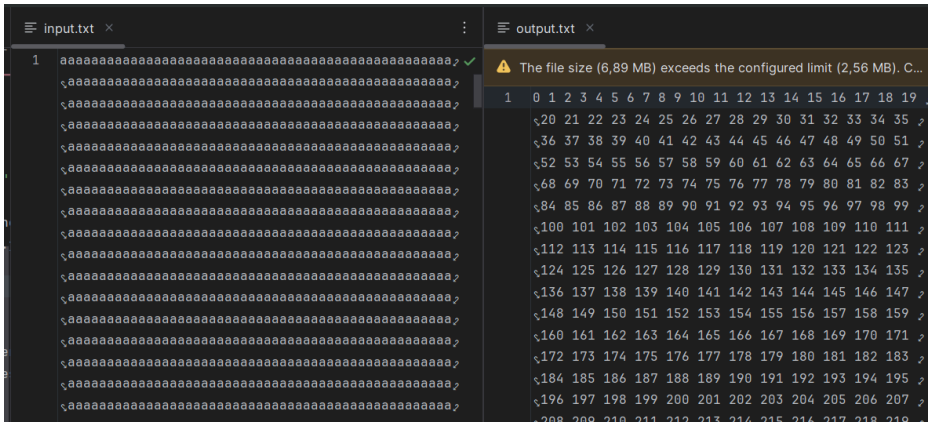
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:





	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0004715	14.87890625
Пример из задачи	0.0004632	14.6640625
Пример из задачи	0.0004465	14.7734375
Верхняя граница диапазона значений входных данных из текста задачи	0.2429344	17.671875

Вывод по задаче:

В данной задаче я реализовала префикс-функцию для строки,которая позволяет эффективно находить повторяющиеся подстроки. Такой алгоритм работает за O(n), что позволило обработать строку длинной 10^6 символов.

Задача №6. Z-функция [1.5 баллов]

Текст задачи:

6 Задача. Z-функция [2 s, 256 Mb, 1.5 балла]

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.** $2 \leq |s| \leq 10^6$.
- **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

Листинг кода:

```
def 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])
        else:  # i > r
            z[i] = 0
               while i + z[i] < n and s[z[i]] == s[i +
z[i]]:
                z[i] += 1
            l, r = i, i + z[i] - 1
    return z


with open("input.txt", "r", encoding='utf-8') as
input_file, open("output.txt", "w", encoding='utf-8')
as output_file:
    s = input_file.readline().strip()
    result = z_function(s)[1:]
```

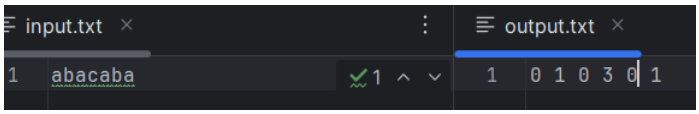
```
output_file.write(' '.join(map(str, result)))
```

Текстовое объяснение решения:

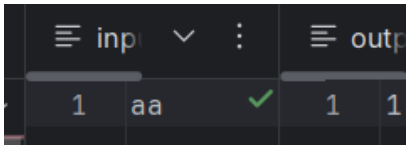
Создаем массив z где каждый элемент z[i] будет длинной наибольшего префикса строки, начинающегося с позиции i. l и r указывают границы префикса. Цикл будет идти не с нулевого символа(тк для него z-функция всегда = длине строки), с первого range(1, n), если i <= r воспользуемся уже подсчитанными значениями, а если i > r (т.е. Вне диапозона) считаем с нуля и обновляем границы.

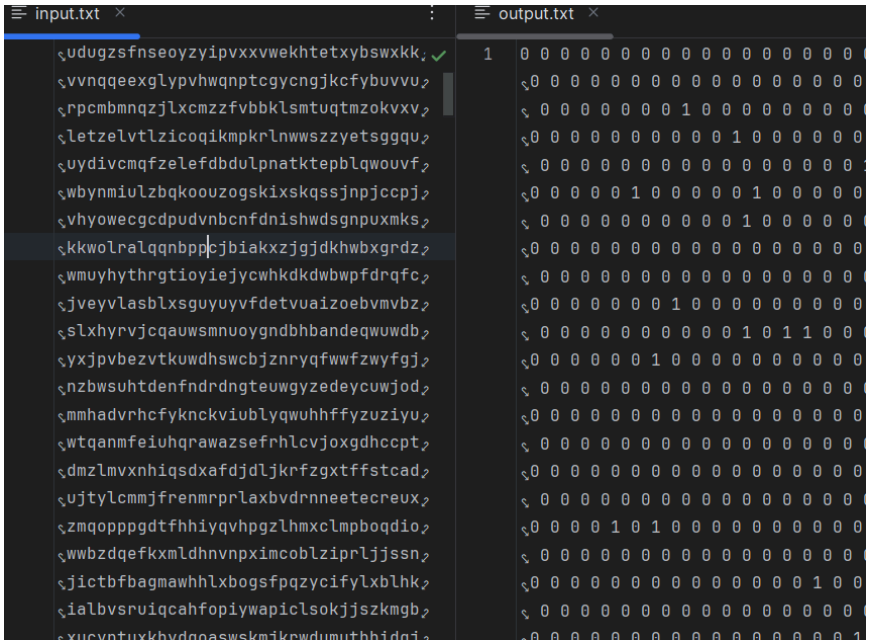
Выводим результат z-функции начиная с первого, а не нулевого.

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	0.0004481	14.87109375
Пример из задачи	: 0.0004687	14.890625
Пример из задачи	0.0004898	14.78515625
Верхняя граница диапазона значений входных данных из текста задачи	0.251642	16.0234375

Вывод по задаче:

Что делает z-функция “на пальцах” вполне понятно, но алгоритм с правильными границами написать оказалось не так очевидно, но все же возможно. Код корректно работает для всех тестов.

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

Текст задачи:

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

Листинг кода:

```
def are_equal(p: str, t: str) -> bool:
    if len(p) != len(t):
        return False
    for i in range(len(p)):
        if p[i] != t[i]:
            return False
    return True


def find_pattern_naive(p: str, t: str):
    res = []
    for i in range(len(t) - len(p) + 1):
        if are_equal(t[i: i + len(p)], p):
            res.append(str(i + 1))
    return res


with open("input.txt", "r", encoding='utf-8') as
input_file, open("output.txt", "w", encoding='utf-8')
as output_file:
    p = input_file.readline().strip()
    t = input_file.readline().strip()
    ans = find_pattern_naive(p, t)
     output_file.write(f'{len(ans)}\n{" ".join(map(str,
ans))}')
```

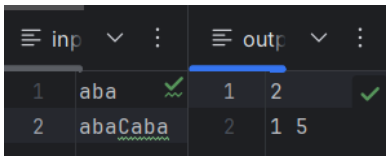
Текстовое объяснение решения:

Проверяем совпадает ли длина строк, если нет сразу возвращаем ложь. Если да дальше проверяем равенство строк посимвольно. Если хоть какие-то 2 символа не равны возвращаем ложь.

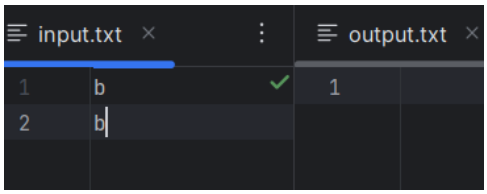
Чтобы найти все позиции вхождения подстроки p в строку t, заводим цикл от 0 до len(t) - len(p). Для каждой позиции i сравнивем стрез строки t[i:i

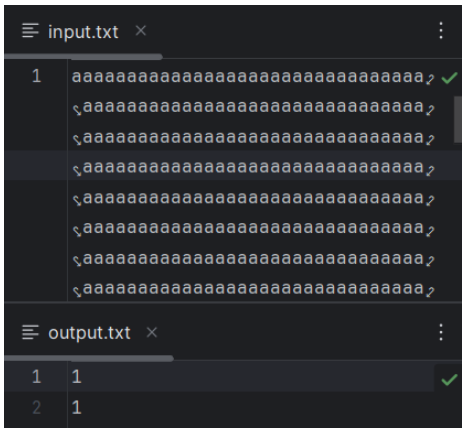
+ len(p)] с подстрокой p с помощью вышеописанной функции. На выходе получаем список с индексами начала вхождения подстроки начиная с 1.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:





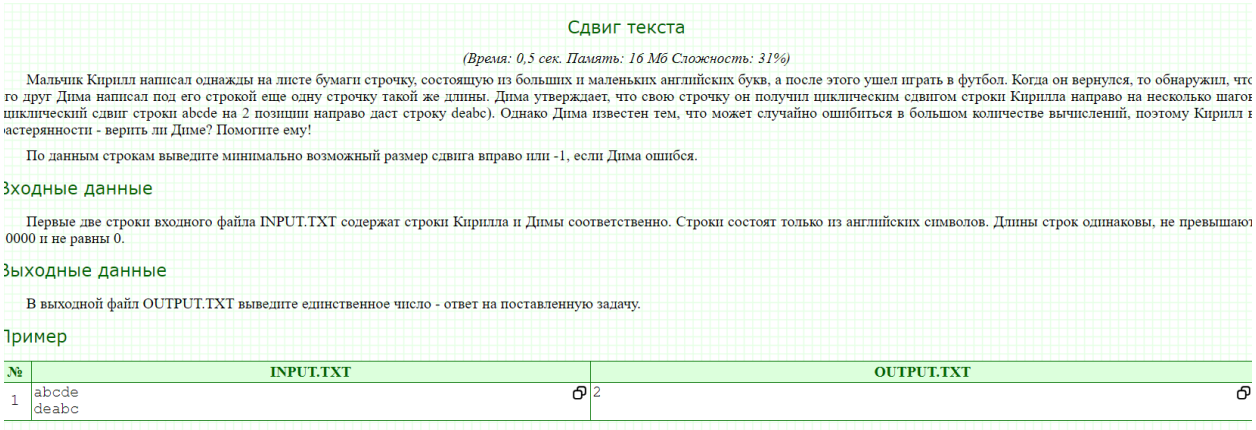
	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0004596	14.65625
Пример из задачи	0.0005054	14.6015625
Верхняя граница диапазона значений входных данных из текста задачи	0.5014624	14.94140625

Вывод по задаче:

Для этой задачи был реализован наивный алгоритм поиска вхождения одной строки в другую. Алгоритм медленный, но подходит для условий задачи с длинной 10**4 символов.

Задача №Д6 . Сдвиг текста

Текст задачи:



Сдвиг текста

(Время: 0,5 сек. Память: 16 Мб Сложность: 31%)

Мальчик Кирилл написал однажды на листе бумаги строчку, состоящую из больших и маленьких английских букв, а после этого ушел играть в футбол. Когда он вернулся, то обнаружил, что его друг Дима написал под его строкой еще одну строчку такой же длины. Дима утверждает, что свою строчку он получил циклическим сдвигом строки Кирилла направо на несколько шагов (циклический сдвиг строки abcde на 2 позиции направо даст строку deabc). Однако Дима известен тем, что может случайно ошибиться в большом количестве вычислений, поэтому Кирилл в растерянности - верить ли Диме? Помогите ему!

По данным строкам выведите минимально возможный размер сдвига вправо или -1, если Дима ошибся.

Входные данные

Первые две строки входного файла INPUT.TXT содержат строки Кирилла и Димы соответственно. Строки состоят только из английских символов. Длины строк одинаковы, не превышают 0000 и не равны 0.

Выходные данные

В выходной файл OUTPUT.TXT выведите единственное число - ответ на поставленную задачу.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	abcde deabc	2

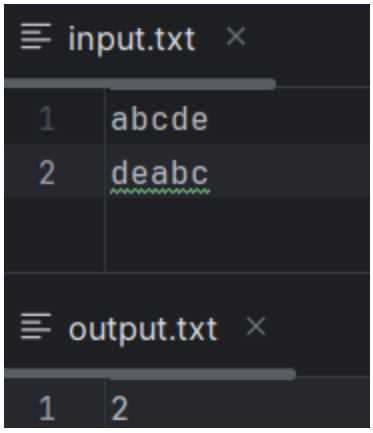
Листинг кода:

```
with open("input.txt", "r", encoding='utf-8') as
input_file, open("output.txt", "w", encoding='utf-8')
as output_file:
    s1 = input_file.readline().strip()
    s2 = input_file.readline().strip()
    k = (s2 + s2).find(s1)
    output_file.write(str(k))
```

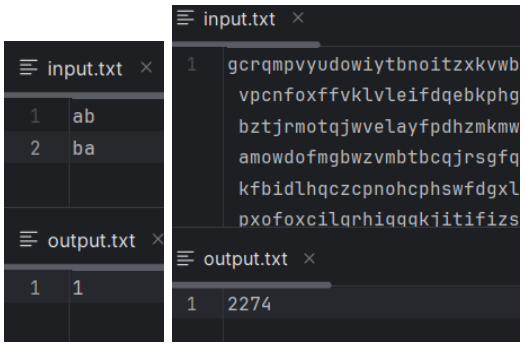
Текстовое объяснение решения:

Находит минимальный циклический сдвиг строки s2, который равен строке s1, используя удвоенную строку s2., записывает индекс этого сдвига в файл

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:



Проверка задачи на acmp:

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
21851963	31.08.2024 21:06:16	Савченко Анастасия Сергеевна	0203	Python	Accepted		0.031	522 Кб

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0004034	14.875
Пример из задачи	0.0004135	14.6328125
Верхняя граница диапазона значений входных данных из текста задачи	0.00055	14.91796875

Вывод по задаче:

Наверное, здесь можно было бы использовать алгоритм КМП, но для этой задачи чтобы пройти все тесты достаточно примитивного подхода с методом find. Пока оставим так, думаю, КМП пригодится для какой-нибудь из следующих задач.

Задача №Д7 . Циклическая строка

Текст задачи:

циклическая строка

(Время: 1 сек. Память: 16 Мб Сложность: 53%)

Строка S была записана много раз подряд, после чего из получившейся строки взяли подстроку и дали Вам. Ваша задача определить минимально возможную длину исходной строки S.

Входные данные

В единственной строке входного файла INPUT.TXT записана строка, которая содержит только английские буквы, длина строки не превышает 50000 символов.

Выходные данные

В выходной файл OUTPUT.TXT нужно вывести одно число - ответ на задачу.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	abababa	2

Листинг кода:

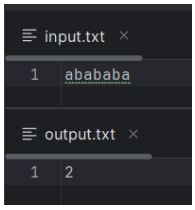
```
def prefix_function(s):
    j = 0
    prefix = [0] * len(s)
    for i in range(1, len(s)):
        j = prefix[i - 1]
        while j > 0 and s[i] != s[j]:
            j = prefix[j - 1]
        if s[i] == s[j]:
            j += 1
        prefix[i] = j
    return prefix


def minimal_length(s):
    n = len(s)
    p = prefix_function(s)
    return n - p[-1]
```

```
with open("input.txt", "r", encoding='utf-8') as
input_file, open("output.txt", "w", encoding='utf-8')
as output_file:
    S = input_file.readline().strip()
    result = minimal_length(S)
    output_file.write(str(result))
```

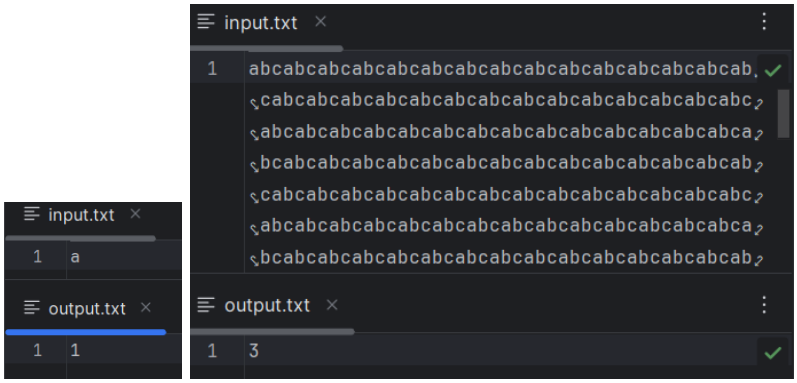
Текстовое объяснение решения:

Используем алгоритм префикс-функции из 5 задачи. p[-1] это значение последнего элемента префикс-функции, а минимальная длина исходной строки S равна n - p[-1].Например, для строки "abababa" префикс-функция: [0, 0, 1, 2, 3, 4, 5]. Длина строки n = 7, n - p[-1] = 2. Действительно "abababa"= 'ab'*4

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:



Проверка задачи на acmp:

№1 - №20

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
21851983	31.08.2024 21:24:57	Савченко Анастасия Сергеевна	0204	Python	Accepted		0.046	2082 Кб

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.0004509	14.8359375

входных данных из текста задачи		
Пример из задачи	0.0004988	14.875
Верхняя граница диапазона значений входных данных из текста задачи	0.013373	16.484375

Вывод по задаче:

В этой простой задаче нашлось еще одно применение префикс-функции.

Задача №Д5 . Поиск подстроки

Текст задачи:

Поиск подстроки

(Время: 0,2 сек. Память: 16 Мб Сложность: 38%)

Найти все вхождения строки T в строке S.

ходные данные

В первой строке входного файла INPUT.TXT записана строка S, во второй строке записана строка T. Обе строки состоят только из английских букв. Длины строк могут быть в диапазоне от 1 о 50 000 включительно.

ыходные данные

В выходной файл OUTPUT.TXT нужно вывести все вхождения строки T в строку S в порядке возрастания. Нумерация позиций строк начинается с нуля.

Пример

№	INPUT.TXT	OUTPUT.TXT
1	ababbababa aba	0 5 7

Листинг кода:

```
def prefix_function(t):
    p = [0] * len(t)
    k = 0
    for i in range(1, len(t)):
        while k > 0 and t[k] != t[i]:
            k = p[k - 1]
        if t[k] == t[i]:
            k += 1
        p[i] = k
    return p
```

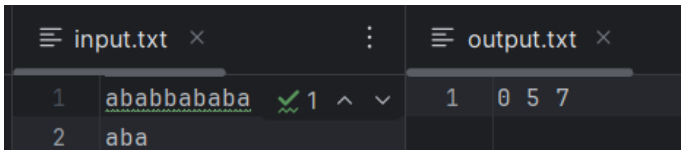
```
def KMP(s, t):
    p = prefix_function(t)
    result = []
    i, j = 0, 0
    while i < len(s):
        if s[i] == t[j]:
            i += 1
            j += 1
        if j == len(t):
            result.append(i - j)
            j = p[j - 1]
        elif i < len(s) and s[i] != t[j]:
            if j != 0:
                j = p[j - 1]
            else:
                i += 1
    return result


with open("input.txt", "r", encoding='utf-8') as input_file, open("output.txt", "w", encoding='utf-8') as output_file:
        s, t = input_file.readline().strip(), input_file.readline().strip()
    result = KMP(s, t)
    output_file.write(" ".join(map(str, result)))
```

Текстовое объяснение решения:

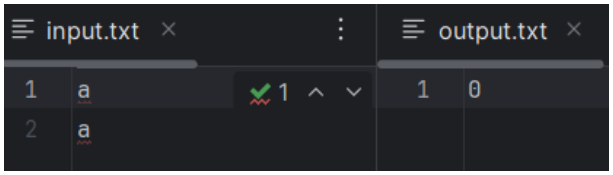
Сначала вычисляется префикс функция. Циклом пробегаем по строке -если символы s[i] и t[j] совпадают, увеличиваем индексы. Если j достигает длины строки t, подстрока найдена добавляем в res индекс вхождения. Для j устанавливаем зн. префикс-функции предыдущего символа.

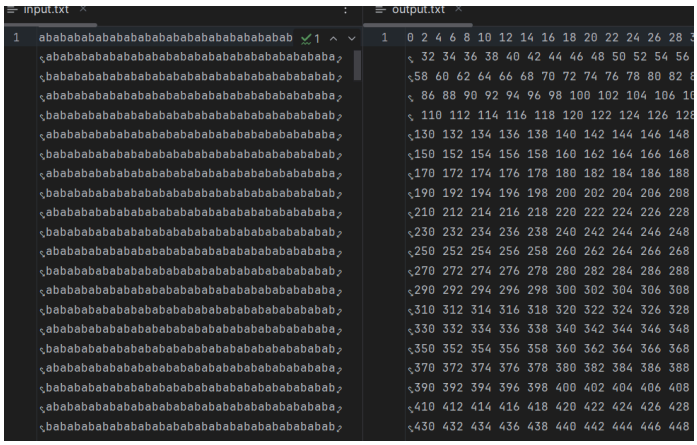
-если символы не совпадают, проверим значение j, если оно не равно нулю, j устанавливается в значение префикс-функции для предыдущего символа, иначе увеличивается индекс i.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:





Проверка задачи на acmp:

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
21852011	31.08.2024 21:49:29	Савченко Анастасия Сергеевна	0202	Python	Accepted		0,093	3486 Кб

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0005713	14.81640625
Пример из задачи	0.0004972	14.7578125
Верхняя граница диапазона значений входных данных из текста задачи	0.0102842	16.609375

Вывод по задаче:

Задача на реализацию алгоритм Кнута-Морриса-Пратта, который эффективен даже для больших текстов, ведь он минимизирует количество сравнений символов при поиске подстроки в строке.

Вывод

В ходе работы над лабораторной я узнала о суффиксах и префиксах строки. Вспомнила хеширование и способы борьбы с коллизиями, а также работу с подстроками. Познакомилась с эффективными алгоритмами поиска паттернов в строке, алгоритмами хэш и z функций, алгоритмом Кнута-Морриса-Пратта. Благода лабораторной работе я узнала как работает быстрый поиск в больших текстах.

Все написанные программы корректно работают даже при больших входных данных, что доказывает эффективность изученных алгоритмов.