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 8

Выполнил:
Макунина А.А.
К32421

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Задачи по варианту

Задача №2. Карта (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.

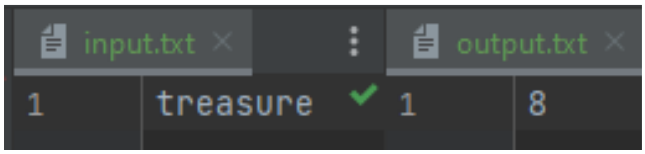
- **Формат ввода / входного файла (input.txt).** В единственной строке входного файла дано послание, написанное на карте.
- **Ограничения на входные данные.** Длина послания не превышает $3 \cdot 10^5$. Гарантируется, что послание может содержать только строчные буквы английского алфавита и пробелы. 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.

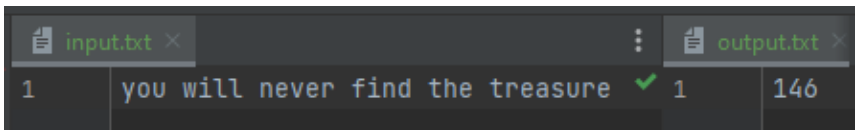
Мы выполняем сортировку путем подсчета по символам и в то же время сохраняем индекс каждого вхождения символа. Затем мы просто смотрим на индексы одних и тех же элементов попарно и вычисляем количество букв между ними.

```
def find_three(line):
    count = 0
    first, last = ord('a'), ord('z')
    full, left = [], []
    for i in range(first, last + 1):
        full.append(0)
        left.append(0)
    for char in line:
        id_ = ord(char)
        full[id_ - first] += 1
    for char in line:
        id_ = ord(char) - first
        left[id_] += 1
        for i in range(len(full)):
            if i == id_:
                right = full[i] - left[i]
```

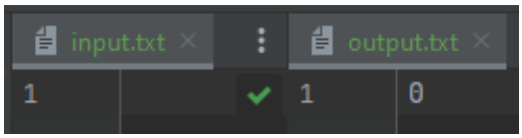
```
                cur_left = left[i] - 1
                count += cur_left * right
            else:
                right = full[i] - left[i]
                count += left[i] * right
    return count


f = open('input.txt')
line = f.readline().split()
t_start = time.perf_counter()
tracemalloc.start()
text = ''
for word in line:
    text += word
w = open('output.txt', 'w')
w.write(str(find_three(text)))
```

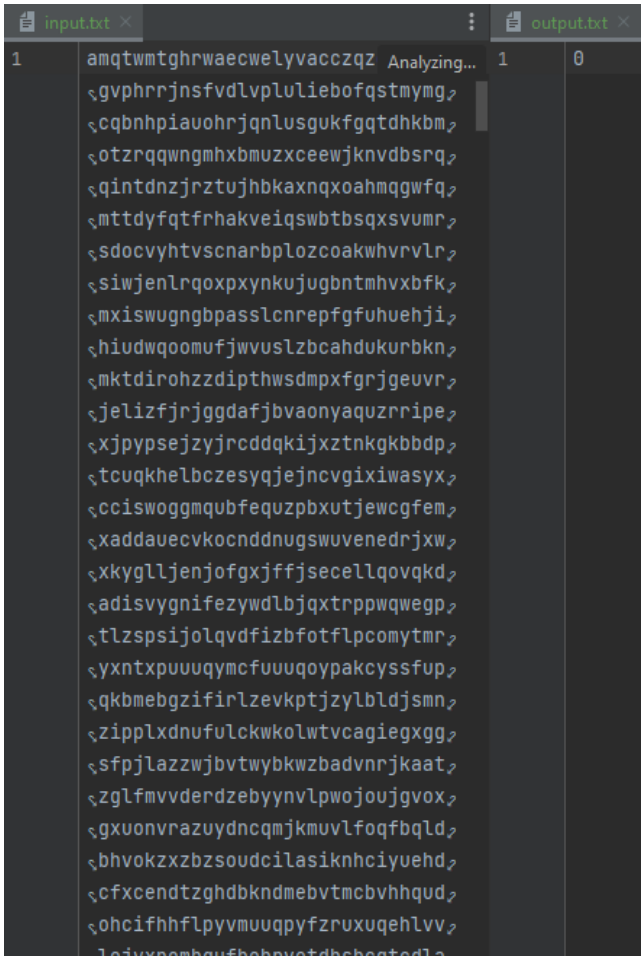
Результат работы кода на примере из текста задачи:





Результат работы кода при минимальном и максимальном значениях:





	Время выполнения, с	Затраты памяти, мб
Нижняя граница диапазона значений входных данных из текста задачи	0.00045869999999999245	0.009421348571777344
Пример из задачи	0.0006171999999999983	0.009497642517089844
Пример из задачи	0.0005522999999999917	0.010630607604980469
Верхняя граница диапазона значений входных данных из текста задачи	0.0006933000000000078	0.009810447692871094

Вывод по задаче: к сожалению, не смогда проверить так как OpenEdu перестало работать. Данный алгоритм работает корректно, так как примеры выполнены успешно!

Задача №5. Префикс-функция (1.5 балла)

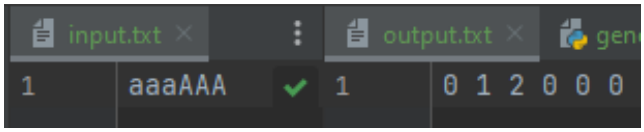
Постройте префикс-функцию для всех непустых префиксов заданной строки s.

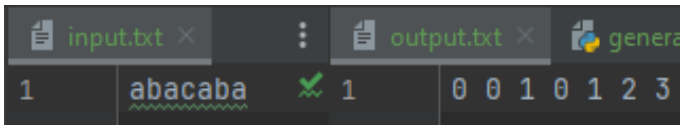
- **Формат ввода / входного файла (input.txt).** Одна строка входного файла содержит s. Строка состоит из букв латинского алфавита.
- **Ограничения на входные данные.** $1 \le |s| \le 10^6$.
- **Формат вывода / выходного файла (output.txt).** Выведите значения префикс-функции для всех префиксов строки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

Рассмотрим три случая: буква суффикса и буква префикса равны – записываем результат идеально, идем дальше; буква суффикса и буква префикса не равны, но до этого они были равны, записываем результат и двигаемся дальше; изначально первая буква префикса не равна первому суффиксу – поэтому результат для такой подстроки равен 0.

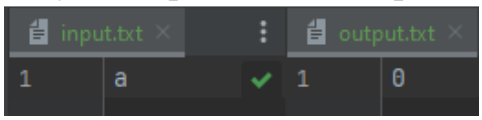
```
f = open('input.txt')
s = f.readline().rstrip()  # возвращает копию строки
с удаленными конечными символами
tracemalloc.start()
t_start = time.perf_counter()
a = [0] * (len(s) + 1)
i, j = 1, 0
while i < len(s):
    if s[i] == s[j]:
        a[i + 1] = j + 1
        i += 1
        j += 1
    else:
        if j > 0:
            j = a[j]
        else:
            a[i + 1] = 0
            i += 1
f.close()
a.pop(0)
w = open('output.txt', 'w')
w.write(' '.join(map(str, a)))
w.close()
```

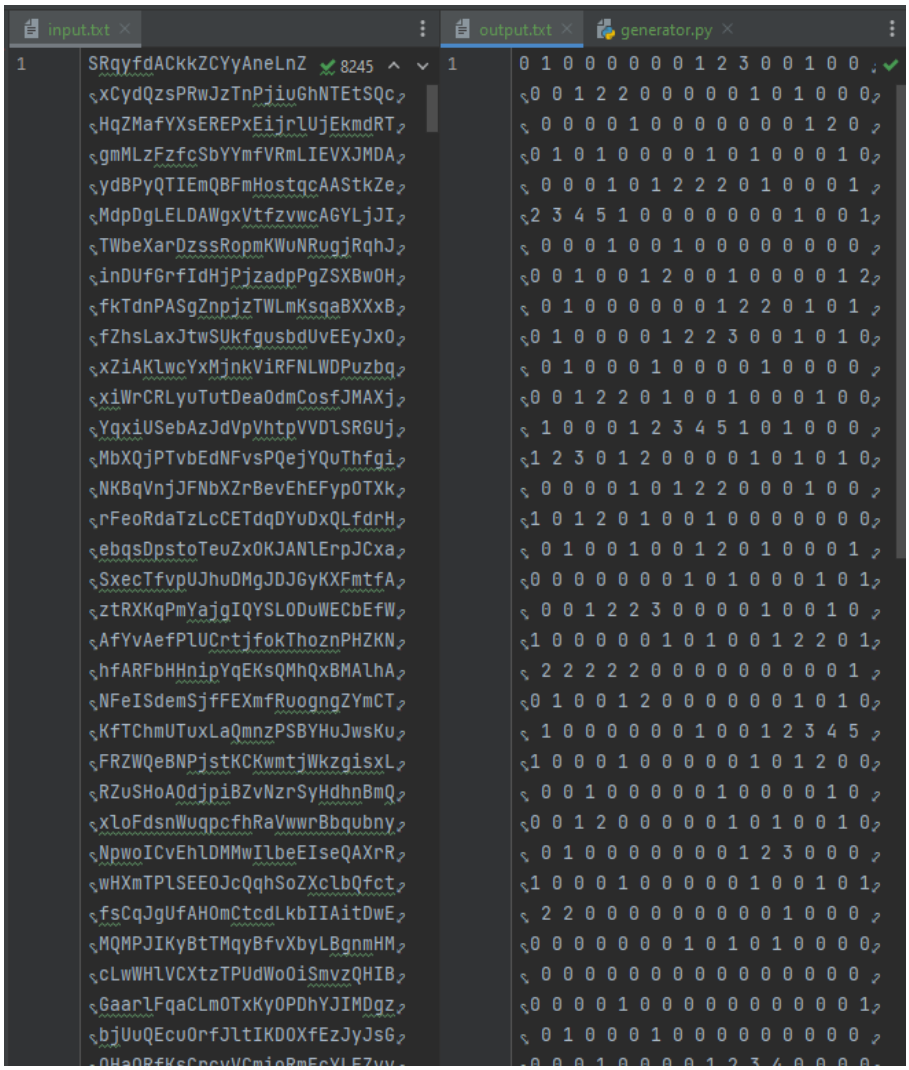
Результат работы кода на примере из текста задачи:





Результат работы кода при минимальном и максимальном значениях:





	Время выполнения, с	Затраты памяти, мб
Нижняя граница диапазона значений входных данных из текста задачи	0.000750600000000004	0.009200096130371094
Пример из задачи	0.0009601999999999944	0.009436607360839844

Пример из задачи	0.0008135999999999977	0.009493827819824219
Верхняя граница диапазона значений входных данных из текста задачи	0.007427400000000084	0.08425426483154297

Вывод по задаче: реализован эффективный алгоритм построения префиксной функции.

Задача №6. Z - функция (1.5 балла)

Постройте Z-функцию для заданной строки s.

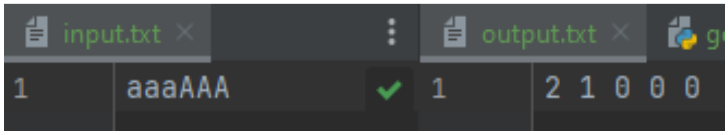
- **Формат ввода / входного файла (input.txt).** Одна строка входного файла содержит s. Строка состоит из букв латинского алфавита.
- **Ограничения на входные данные.** $2 \le |s| \le 10^6$.
- **Формат вывода / выходного файла (output.txt).** Выведите значения Z-функции для всех индексов $1, 2, ...,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

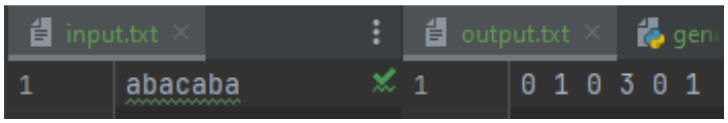
Этот алгоритм является оптимизацией наивного алгоритма вычисления z-функции. Мы сохраним указатель r на место, где функция z уже была вычислена (правый край). Если мы вышли за пределы правого края, то мы рассматриваем функцию z. Если край того, что мы уже вычислили, больше текущего индекса, что означает, что мы берем готовый результат из массива результатов. Если r - наша текущая граница, что означает, то мы вычислям для него функцию z, используя результаты предыдущих.

```
f = open('input.txt')
s = f.readline().rstrip()
t_start = time.perf_counter()
tracemalloc.start()
f.close()
res = [0] * len(s)
ln = len(s)
l, r, n = 0, 0, 0
for i in range(1, ln):
    if i > r:
        l, r = i, i
        while r < ln and s[r-l] == s[r]:
```

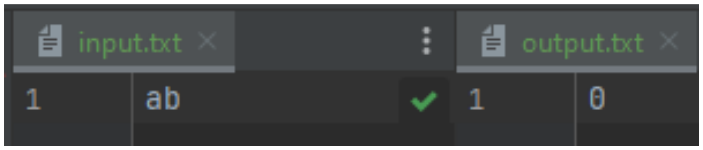
```
            r += 1
        res[i] = r-l
        r -= 1
    else:
        n = i - l
        if res[n] < r - i + 1:
            res[i] = res[n]
        else:
            l = i
            while r < ln and s[r-l] == s[r]:
                r += 1
            res[i] = r - l
            r -= 1
res.pop(0)
last = res.pop(-1)
w = open('output.txt', 'w')
for elem in res:
    w.write(str(elem) + ' ')
w.write(str(last))
w.close()
```

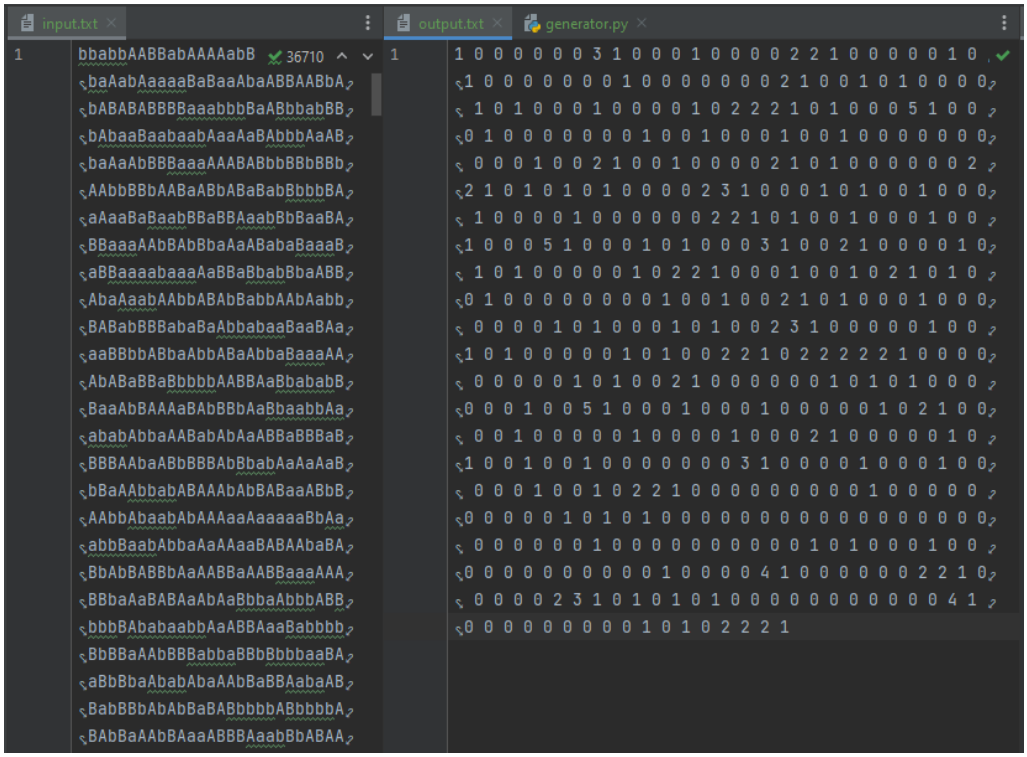
Результат работы кода на примере из текста задачи:





Результат работы кода при минимальном и максимальном значениях:





	Время выполнения, с	Затраты памяти, мб
Нижняя граница диапазона значений входных данных из текста задачи	0.0007781999999999928	0.010245323181152344
Пример из задачи	0.0010780999999999985	0.010291099548339844
Пример из задачи	0.0007858000000000032	0.010323524475097656
Верхняя граница диапазона значений входных данных из текста задачи	0.008490399999999898	0.03905200958251953

Вывод по задаче: код работает исправно судя по примерам!

Дополнительные задачи

Задача №1. Наивный поиск подстроки в строке (1 балл)

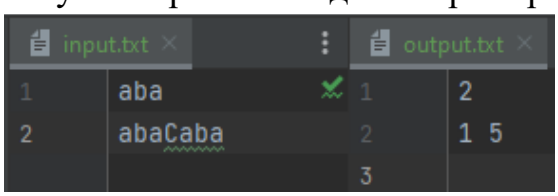
Даны строки p и t. Требуется найти все вхождения строки p в строку t в качестве подстроки.

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

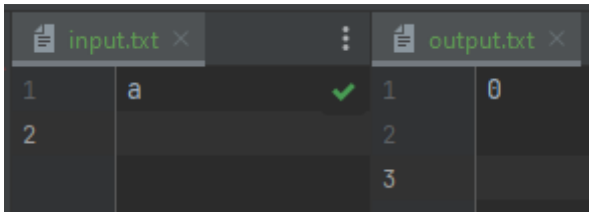
В первой части читаются первые две строки из файла. Затем, используя циклы, мы находим вхождение первого символа строки p в строку t. Если мы получаем и не выходим за пределы строки t, то проверяем равенство со строкой p срезом от i до i+len(p). Если все совпадает, то мы добавляем длину найденной строки к параметру цикла i и добавляем индекс к результату.

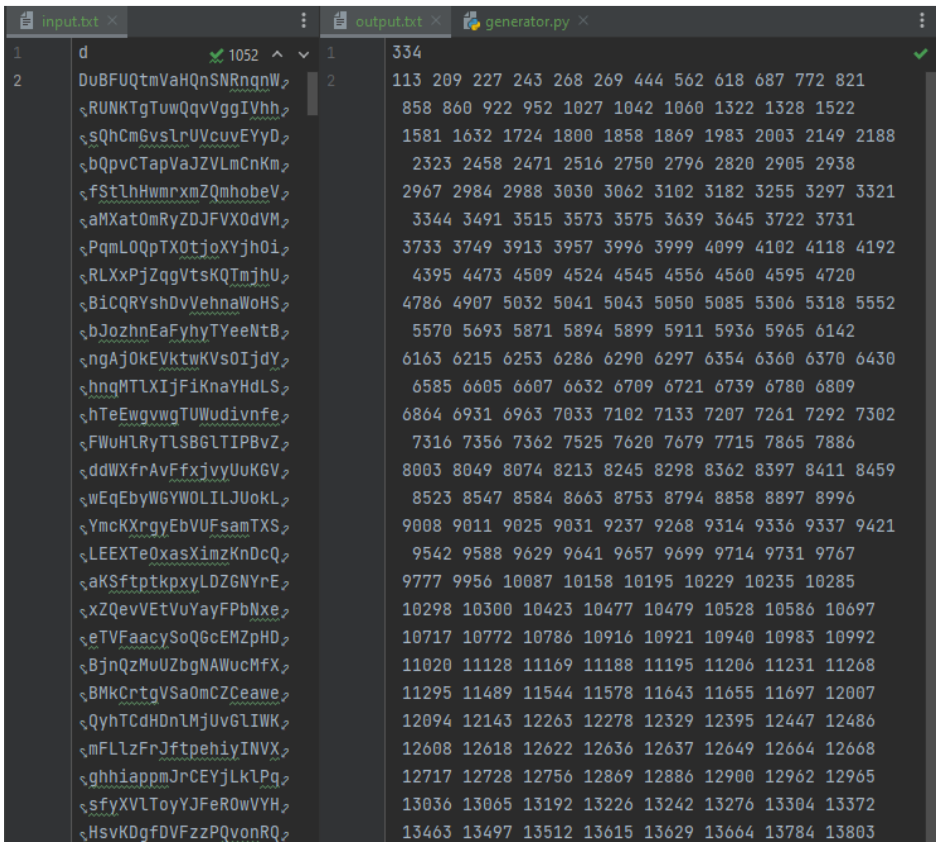
```
f = open('input.txt')
p = f.readline().rstrip()
t = f.readline().rstrip()
f.close()
t_start = time.perf_counter()
tracemalloc.start()
result = []
for i in range(len(t)):
    if p[0] == t[i] and len(p) + i <= len(t):
        if t[i: i + len(p)] == p:
            result.append(str(i + 1))
            i += len(p)
w = open('output.txt', 'w')
w.write(str(len(result)) + '\n')
w.write(str(' '.join(result)) + '\n')
```

Результат работы коды на примерах из задачи:



Результат работы кода при минимальном и максимальном значениях:





	Время выполнения, с	Затраты памяти, мб
Нижняя граница диапазона значений входных данных из текста задачи	0.0004315999999999903	0.009119987487792969
Пример из задачи	0.0004269000000000009	0.009248733520507812
Верхняя граница диапазона значений входных данных из текста задачи	0.012720599999999999	0.03367042541503906

Вывод по задаче: как в егэ…

Задача №3. Паттерн в тексте (1 балл)

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

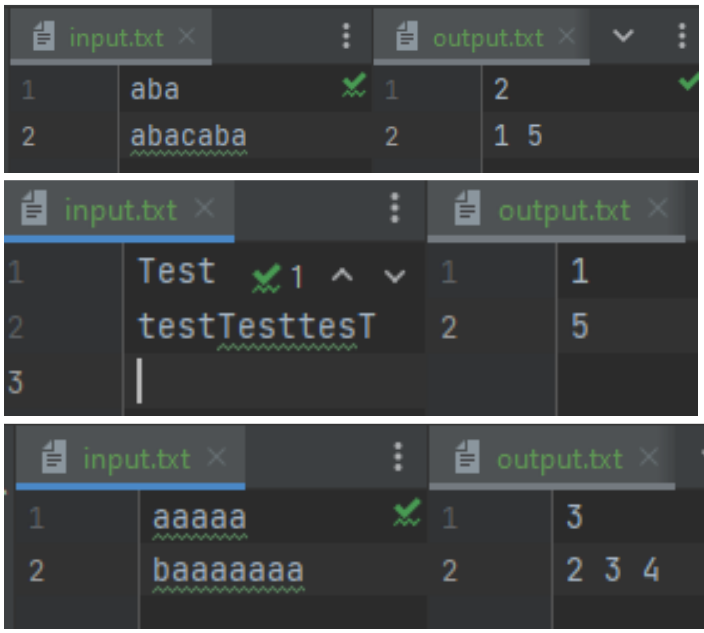
- **Формат ввода / входного файла (input.txt).** На входе две строки: паттерн P и текст T. Требуется найти все вхождения строки P в строку T в качестве подстроки.
- **Ограничения на входные данные.** $1 \leq |P|, |T| \leq 10^6$. Паттерн и текст содержат только латинские буквы.
- **Формат вывода / выходного 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.

В этой задаче мне нужно было реализовать алгоритм Рабина-Карпа для поиска заданного шаблона в заданном тексте. Алгоритм Рабина-Карпа заключается в том, что он использует хеширование для поиска шаблона. Вот для чего я использовала функции хеширования.

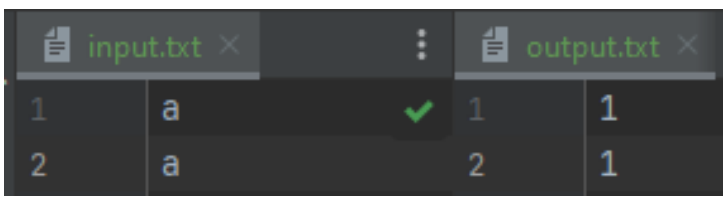
```
def PolyHash(pat, p, x):
    res = 0
    for i in reversed(range(lenp)):
        res = (res * x + ord(pat[i])) % p
    return res % p


def CalculateHashes(T, p, x):
    global lenp, lens
    H = [0] * (lens - lenp + 1)
    S = T[lens - lenp: lens]
    H[lens - lenp] = PolyHash(S, p, x)
    y = 1
    for i in range(1, lenp + 1):
        y = (y * x) % p
    for i in range(lens - lenp - 1, -1, -1):
        H[i] = (x * H[i + 1] + ord(T[i]) - y *
ord(T[i + lenp]) + p) % p
    return H


def Rabin_Karp(pattern, string):
    global lenp, lens
    p = 10 ** 9 + 9
    x = 127
    count = 0
```

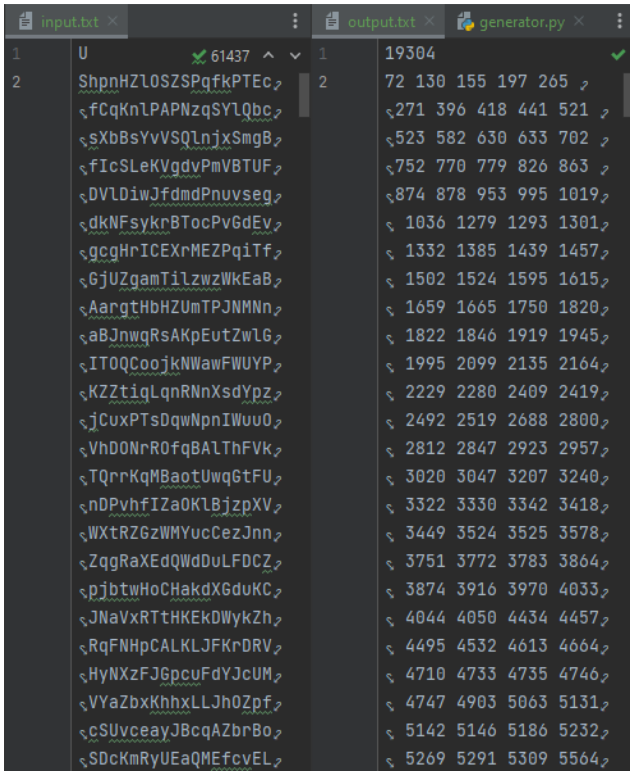
```
    res = []
    hash_pat = PolyHash(pattern, p, x)
    hash_str = CalculateHashes(string, p, x)
    for i in range(lens - lenp + 1):
        if hash_pat != hash_str[i]:
            continue
        count += 1
        res.append(str(i + 1))
    w = open('output.txt', 'w')
    w.write(str(count) + "\n" + " ".join(res))


f = open('input.txt')
pattern = f.readline().rstrip()
string = f.readline()
lenp, lens = len(pattern), len(string)
Rabin_Karp(pattern, string)
```

Результат работы коды на примерах из задачи:



Результат работы кода при минимальном и максимальном значениях:





	Время выполнения, с	Затраты памяти, мб
Нижняя граница диапазона значений входных данных из текста задачи	0.0006895000000000026	0.0107574462890625
Пример из задачи	0.0009308999999999984	0.010981559753417969
Пример из задачи	0.0008914999999999895	0.010860443115234375
Пример из задачи	0.0009713999999999973	0.010863304138183594
Верхняя граница диапазона значений входных данных из текста задачи	2.3556208000000005	10.1099853515625

Вывод по задаче: немного не уложилась в верхнюю границу диапазона значений, но ничего страшного! В целом оцениваю работу на хорошо!

Задача №4. Равенство подстрок (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.

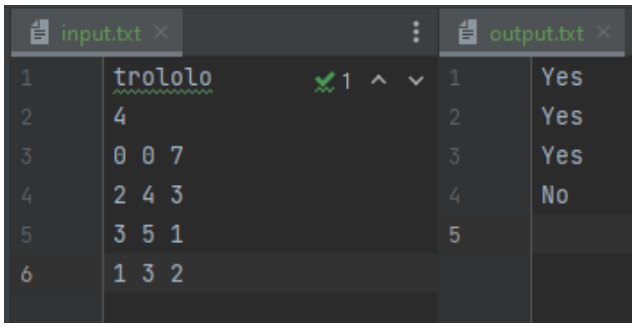
- **Формат ввода / входного файла (input.txt).** Первая строка содержит строку s, состоящую из строчных латинских букв. Вторая строка содержит количество запросов q. Каждая из следующих q строк задает запрос тремя целыми числами a, b и l.
- **Ограничения на входные данные.** $1 \le |s| \le 500000, 1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

Для решения создается функция hash_function, в которой для переданной строки вычисляются хэши для каждого символа, при этом хэши для каждого символа вычисляются двумя способами - путем деления на $10^9 + 7$ и $10^9 + 9$. Записываем строку в переменную string и передаём ее функции hash_function, пишем результат функции в переменную hash. Мы перебираем все строки, которые необходимо проверить, и сравниваем их хэши. Если они совпадают, пишем в файл «Yes», в противном случае - «No».

```
def hash_function(string):
    a = 12
    num_1 = num_2 = 10 ** 9 + 7
    n = len(string) + 1
    hashes = [(0, 0)] * n
    for i in range(1, n):
        number = ord(string[i - 1])
        hash_1 = (a * hashes[i - 1][0] + number) %
num_1
        hash_2 = (a * hashes[i - 1][0] + number) %
num_2
        hashes[i] = (hash_1, hash_2)

    return hashes


f = open('input.txt')
string = f.readline().rstrip()
t_start = time.perf_counter()
tracemalloc.start()
```

```
hash = hash_function(string)
n = int(f.readline())
w = open('output.txt', 'w')
for i in range(n):
    (ind_1, ind_2, len) = map(int,
f.readline().split())
    str1_hash1 = (hash[ind_1 + len][0] - (12 ** len)
* hash[ind_1][0]) % (10 ** 9 + 7)
    str2_hash1 = (hash[ind_2 + len][0] - (12 ** len)
* hash[ind_2][0]) % (10 ** 9 + 7)
    str1_hash2 = (hash[ind_1 + len][1] - (12 ** len)
* hash[ind_1][1]) % (10 ** 9 + 9)
    str2_hash2 = (hash[ind_2 + len][1] - (12 ** len)
* hash[ind_2][1]) % (10 ** 9 + 9)
    if str1_hash1 == str1_hash2 and str1_hash2 ==
str2_hash2:
        w.write('Yes\n')
    else:
        w.write('No\n')
f.close()
w.close()
```

Результат работы кода:



	Время выполнения, с	Затраты памяти, мб
Пример из задачи	0.0009054000000000006	0.018619537353515625

Вывод по задаче: успешно и нетрудно!

Задача №7. Наибольшая общая подстрока (2 балла)

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.

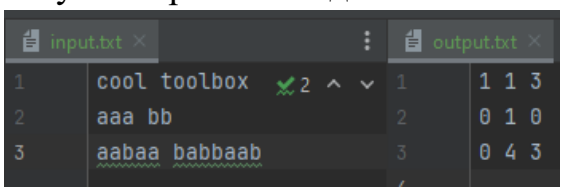
- **Формат ввода / входного файла (input.txt).** Каждая строка входных данных содержит две строки s и t, состоящие из строчных латинских букв.
- **Ограничения на входные данные.** Суммарная длина всех s, а также суммарная длина всех s не превышает 100 000.
- **Формат вывода / выходного файла (output.txt).** 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.

Реализовано с использованием хэширования для поиска самой длинной общей подстроки используется двоичный поиск.

```
def PolyHash(P, l, p, x):
    res = 0
    for i in reversed(range(l)):
        res = (res * x + ord(P[i])) % p
    return res % p


def CalculateHashes(T, l, k, p, x):
    H = [0] * (l - k + 1)
    S = T[l - k: l]
    H[l - k] = PolyHash(S, k, p, x)
    y = 1
    for i in range(1, k + 1):
        y = (y * x) % p
    for i in range(l - k - 1, -1, -1):
        H[i] = (x * H[i + 1] + ord(T[i]) - y *
ord(T[i + k]) + p) % p
    return H


f = open('input.txt')
w = open('output.txt', 'w')
```

```
t_start = time.perf_counter()
tracemalloc.start()
while True:
    line = f.readline()
    if not line:
        exit()
    s, t = map(str, line.split())
    lS, lT = len(s), len(t)
    k = min(lS, lT)
    p = 10 ** 9 + 7
    x = random.randint(1, p - 1)
    flag = False
    for i in reversed(range(1, k + 1)):
        Hs = CalculateHashes(s, lS, i, p, x)
        Ht = CalculateHashes(t, lT, i, p, x)
        for j in range(len(Hs)):
            for h in range(len(Ht)):
                if Hs[j] == Ht[h]:
                    w.write(str(j) + ' ' + str(h) + '
' + str(i) + '\n')
                    flag = True
                    break
            if flag:
                break
        if flag:
            break
    if not flag:
        w.write('0' + ' ' + '1' + ' ' + '0' + '\n')
f.close()
w.close()
```

Результат работы кода:



	Время выполнения, с	Затраты памяти, мб
Пример из задачи	0.0006526000000000032	0.011708259582519531

Вывод по задаче: спасибо хэшированию, всё успешно

Задача №9. Декомпозиция строки (2 балла)

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

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

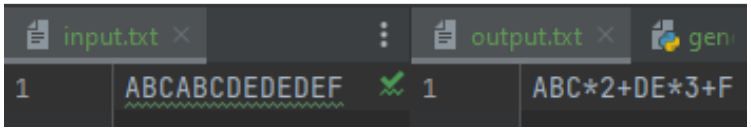
- **Формат ввода / входного файла (input.txt).** Одна строка входного файла содержит s. Строка состоит из букв латинского алфавита.
- **Ограничения на входные данные.** $1 \le |s| \le 5 \cdot 10^3$.
- **Формат вывода / выходного файла (output.txt).** Выведите оптимальное представление строки, данной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

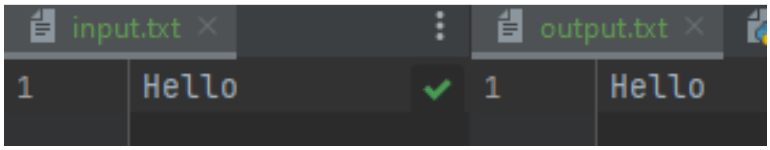
В этой задаче мне нужно было реализовать декомпозицию строк. Реализовано с использованием z-функции. Затем я прошлась по строке и декомпозировала ее.

```
def z_func(s):
    n = len(s)
    z = [0] * n
    l = r = 0
    for i in range(1, n):
        if i <= r:
            z[i] = min(z[i - l], r - i + 1)
        while i + z[i] < n and s[i + z[i]] ==
s[z[i]]:
            z[i] += 1
        if i + z[i] > r:
            l, r = i, i + z[i] - 1
    return z


def step(ln, k, prev):
    res = k + 2 + len(str(ln // k))
    if ln == k:
        res -= 2
    if prev == n:
        res -= 1
    return res
```

```
f = open('input.txt')
s = f.readline()
t_start = time.perf_counter()
tracemalloc.start()
n = len(s)
s += '_'
dp = [n - i for i in range(n + 1)]
to = [[n - i, n - i] for i in range(n + 1)]
for i in range(n - 2, -1, -1):
    z = z_func(s[i:])
    if dp[i] > dp[i + 1] + 2:
        dp[i] = dp[i + 1] + 2
        to[i] = [1, 1]
    for j in range(i + 1, n + 1):
        k = 1
        while k * k <= j - i:
            if (j - i) % k:
                k += 1
                continue
            if z[k] + k >= j - i:
                if dp[i] > dp[j] + step(j - i, k, j):
                    dp[i] = dp[j] + step(j - i, k, j)
                    to[i] = [j - i, k]
            if z[(j - i) // k] + (j - i) // k >= j -
i:
                if dp[i] > dp[j] + step(j - i, (j -
i) // k, j):
                    dp[i] = dp[j] + step(j - i, (j -
i) // k, j)
                    to[i] = [j - i, (j - i) // k]
            k += 1

w = open('output.txt', 'w')
i = 0
while i < n:
    if i > 0:
        w.writelines('+')
    w.writelines(s[i: i + to[i][1]])
    if to[i][0] != to[i][1]:
        w.writelines('*' + str(to[i][0] // to[i][1]))
    i += to[i][0]
```

Результат работы кода на примерах из текста задачи:





Результат работы кода при минимальном и максимальном значениях:

	Время выполнения, с	Затраты памяти, мб
Нижняя граница диапазона значений входных данных из текста задачи	0.0004378999999999911	0.009369850158691406
Пример из задачи	0.0008195000000000008	0.013079643249511719
Пример из задачи	0.0005897000000000124	0.012104988098144531

Вывод по задаче: хорошая задача, но не удалось удачно запустить тест на максимальную границу.

Дополнительная задача №3. Имена

(Время: 1 сек. Память: 16 Мб Сложность: 38%)

На далекой планете Тау Кита есть непонятные нам обычаи. Например, таукитяне очень необычно для землян выбирают имена своим детям. Родители так выбирают имя ребенку, чтобы оно могло быть получено как удалением некоторого набора букв из имени отца, так и удалением некоторого набора букв из имени матери. Например, если отца зовут «abacaba», а мать — «bbccaa», то их ребенок может носить имена «a», «bba», «bcaa», но не может носить имена «aaa», «ab» или «bbc». Возможно, что имя ребенка совпадает с именем отца и/или матери, если оно может быть получено из имени другого родителя удалением нескольких (возможно, ни одной) букв.

Пусть отец по имени X и мать по имени Y выбирают имя своему новорожденному ребенку. Так как в таукитянских школах учеников часто вызывают к доске в лексикографическом порядке имен учеников, то есть в порядке следования имен в словаре, то они хотят выбрать своему ребенку такое имя, чтобы оно лексикографически следовало как можно позже.

Формально, строка S лексикографически больше строки T, если выполняется одно из двух условий:

• строка T получается из S удалением одной или более букв с конца строки S;

• первые (i - 1) символов строк T и S не различаются, а буква в i-й позиции строки T следует в алфавите раньше буквы в i-й позиции строки S.

Требуется написать программу, которая по именам отца и матери находит лексикографически наибольшее имя для их ребенка.

Входные данные

Первая строка входного файла INPUT.TXT содержит имя отца X. Вторая строка входного файла содержит имя матери Y. Каждое имя состоит из строчных букв английского алфавита, включает хотя бы одну букву и имеет длину не более 10^5 букв.

Выходные данные

В выходной файл OUTPUT.TXT выведите искомое лексикографически наибольшее из возможных имен ребенка. В случае, если подходящего имени для ребенка не существует, выходной файл должен быть пустым.

Мы подсчитываем количество одинаковых букв в именах родителей, затем собираем из них имя будущего ребенка в правильном порядке.

```
f = open('input.txt')
r = ''
mam = f.readline()
dad = f.readline()
f.close()
words = list(set(mam) & set(dad))
if len(words) != 0:
    words.sort()
    words.reverse()
    while len(words) != 0:
        g = min(mam.count(words[0]),
dad.count(words[0]))
        r += words[0] * g
        for i in range(g):
            mam = mam[mam.find(words[0]) + 1:]
            dad = dad[dad.find(words[0]) + 1:]
        words.pop(0)
```

```
w = open('output.txt', 'w')
w.write(r)
w.close()
```

Результат работы кода:

Тест	Результат	Время	Память
1	Accepted	0,015	350 Кб
2	Accepted	0,031	338 Кб
3	Accepted	0,015	350 Кб
4	Accepted	0,031	446 Кб
5	Accepted	0,015	342 Кб
6	Accepted	0,015	342 Кб
7	Accepted	0,031	346 Кб
8	Accepted	0,015	338 Кб
9	Accepted	0,015	342 Кб
10	Accepted	0,015	346 Кб
11	Accepted	0,015	662 Кб
12	Accepted	0,14	638 Кб
13	Accepted	0,14	570 Кб
14	Accepted	0,937	1290 Кб
15	Accepted	0,093	638 Кб
16	Accepted	0,328	746 Кб
17	Accepted	0,171	730 Кб
18	Accepted	0,375	738 Кб
19	Accepted	0,39	758 Кб
20	Accepted	0,328	778 Кб
21	Accepted	0,015	346 Кб
22	Accepted	0,031	338 Кб
23	Accepted	0,031	338 Кб
24	Accepted	0,031	426 Кб
25	Accepted	0,015	342 Кб
26	Accepted	0,031	346 Кб
27	Accepted	0,015	350 Кб
28	Accepted	0,015	350 Кб
29	Accepted	0,015	342 Кб
30	Accepted	0,031	342 Кб
31	Accepted	0,031	666 Кб
32	Accepted	0,046	518 Кб
33	Accepted	0,031	630 Кб
34	Accepted	0,796	846 Кб
35	Accepted	0,015	614 Кб
36	Accepted	0,015	594 Кб
37	Accepted	0,515	750 Кб
38	Accepted	0,031	730 Кб
39	Accepted	0,046	762 Кб
40	Accepted	0,046	662 Кб
41	Accepted	0,031	546 Кб
42	Accepted	0,031	502 Кб
43	Accepted	0,015	486 Кб
44	Accepted	0,5	738 Кб
45	Accepted	0,015	582 Кб
46	Accepted	0,031	646 Кб
47	Accepted	0,203	1798 Кб
48	Accepted	0,062	542 Кб
49	Accepted	0,015	530 Кб
50	Accepted	0,031	538 Кб

Вывод по задаче: алгоритм использует структуру данных python – set, с ее помощью вычисляется количество уникальных букв в именах мамы и папы.

Дополнительная задача №4. Суффиксы

(Время: 1 сек. Память: 16 Мб Сложность: 45%)

Назовем строкой последовательность из маленьких букв английского алфавита. Строкой, например, является пустая последовательность "", слово "aabaf" или бесконечная последовательность букв "a".

i-ый суффикс S_i строки S – это строка S, из которой вырезаны первые i букв: так, для строки S = "aabaf" суффиксы будут такими:

S_0 = "aabaf"

S_1 = "abaf"

S_2 = "baf"

S_3 = "af"

S_4 = "f"

$S_5 = S_6 = S_7 = \ldots =$ ""

Суффиксы определены для всех $i > 0$.

Циклическое расширение S^* конечной строки S – это строка, полученная приписыванием ее к самой себе бесконечное количество раз. Так,

$S^* = S_0^*$ = "aabafaabafaa..."

S_1^* = "abafabafabaf..."

S_2^* = "bafbafbafbaf..."

S_3^* = "afafafafafaf..."

S_4^* = "ffffffffff..."

$S_5^* = S_6^* = S_7^* = \ldots =$ ""

По данной строке S выясните, сколько ее суффиксов S_i имеют такое же циклическое расширение, как и сама строка S, то есть количество таких i, что $S^* = S_i^*$.

Входные данные

Входной файл INPUT.TXT содержит строку S, состоящую не менее, чем из одной и не более, чем из 100 000 маленьких английских букв.

Выходные данные

В выходной файл OUTPUT.TXT выведите количество суффиксов строки S, имеющих такое же циклическое расширение, как и она сама.

В этой задаче у нас есть два варианта: либо такой подстрокой является S0, то есть сама строка, этот вариант обрабатывается в блоке elif, либо строка состоит из повторяющихся шаблонов, тогда мы получаем результат через блок if.

```
s = input()
n = 1
l1 = l2 = len(s)
while n * n <= l2:
    if l2 % n == 0:
        c = l2 // n
        if s[:n] * c == s:
```

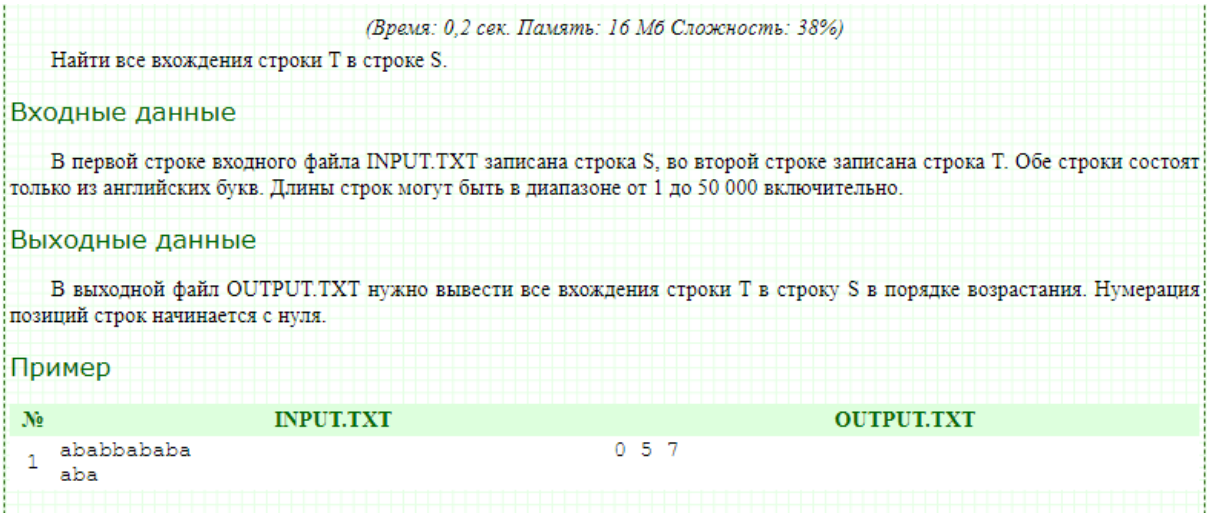
```
            l1 = n
            break
        elif s[:c] * n == s:
            l1 = c
    n += 1
print(len(s) // l1)
```

Результат работы кода:

Тест	Результат	Время	Память
1	Accepted	0,046	342 Кб
2	Accepted	0,015	350 Кб
3	Accepted	0,015	338 Кб
4	Accepted	0,015	342 Кб
5	Accepted	0,031	354 Кб
6	Accepted	0,015	342 Кб
7	Accepted	0,031	346 Кб
8	Accepted	0,031	642 Кб
9	Accepted	0,015	642 Кб
10	Accepted	0,015	642 Кб
11	Accepted	0,015	638 Кб
12	Accepted	0,031	646 Кб
13	Accepted	0,015	638 Кб
14	Accepted	0,015	646 Кб
15	Accepted	0,031	638 Кб
16	Accepted	0,031	638 Кб
17	Accepted	0,015	346 Кб
18	Accepted	0,015	602 Кб
19	Accepted	0,046	610 Кб
20	Accepted	0,031	646 Кб
21	Accepted	0,015	530 Кб
22	Accepted	0,031	534 Кб
23	Accepted	0,031	534 Кб
24	Accepted	0,031	542 Кб
25	Accepted	0,046	606 Кб
26	Accepted	0,031	566 Кб
27	Accepted	0,015	554 Кб
28	Accepted	0,015	562 Кб
29	Accepted	0,015	550 Кб
30	Accepted	0,031	602 Кб
31	Accepted	0,015	530 Кб
32	Accepted	0,046	638 Кб
33	Accepted	0,015	610 Кб
34	Accepted	0,015	606 Кб
35	Accepted	0,015	570 Кб
36	Accepted	0,015	566 Кб
37	Accepted	0,031	558 Кб
38	Accepted	0,015	550 Кб
39	Accepted	0,015	598 Кб
40	Accepted	0,031	542 Кб
41	Accepted	0,015	642 Кб
42	Accepted	0,046	602 Кб

Вывод по задаче: алгоритм основан на предположении, что не существует такой строки, циклическое расширение которой равно циклическому расширению подстроки, в то время как подстрока * k != строке.

Дополнительная задача №5. Поиск подстроки



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

Найти все вхождения строки T в строке S.

Входные 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 до 50 000 включительно.

Выходные 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

Мы берем p-функцию и используем ее для решения. Мы берем строку, которую мы ищем, и ставим разделитель после нее ‘ ‘ и саму строку, в которой мы будем искать. После прохождения функции по всему массиву будет сохранена часть, описывающая все, что после ‘ ‘ информация о том, где есть нужные нам вхождения. Поскольку мы знаем длину строки, которую мы ищем, мы можем буквально видеть, где находится эта длина, и это будут наши индексы. Перед выводом нужно вычесть длину подстроки, которую ищем.

```
def find_subline(line, fin_len):
    stripped_line = list(line)
    len_line = len(line)
    pref = [0 for i in range(len_line)]
    j = 0

    for i in range(2, len_line + 1):
        while j > 0 and stripped_line[j] !=
stripped_line[i - 1]:
            j = pref[j - 1]
        if stripped_line[j] == stripped_line[i - 1]:
            j += 1
```

```
            pref[i - 1] = j
        _ = ''
        for i in range(len(pref)):
            if pref[i] == fin_len:
                _ += str(i - 2 * fin_len) + ' '
        return _


f = open('input.txt')
line = f.readline().strip()
to_find = f.readline().strip()
fin_len = len(to_find)
mix = to_find + ' ' + line
_ = find_subline(mix, fin_len)
f.close()
w = open('output.txt', 'w')
w.write(_.strip())
w.close
```

Результат работы кода:

Тест	Результат	Время	Память
1	Accepted	0,015	346 Кб
2	Accepted	0,031	354 Кб
3	Accepted	0,015	354 Кб
4	Accepted	0,015	350 Кб
5	Accepted	0,015	338 Кб
6	Accepted	0,015	346 Кб
7	Accepted	0,015	1266 Кб
8	Accepted	0,046	1398 Кб
9	Accepted	0,046	1442 Кб
10	Accepted	0,093	6 Мб
11	Accepted	0,109	4550 Кб
12	Accepted	0,062	4554 Кб
13	Accepted	0,015	350 Кб

Вывод по задаче: тесты пройдены успешно и хорошо

Дополнительная задача №6. 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 10000 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

Повторно «приклеиваем» первую строку, каждый раз сдвигая ее на единицу вправо, если мы сдвинули строку n раз, n равно длине строки, но она так и не стала равной второй строке, тогда все в порядке.

```
f = open('input.txt')
s1 = f.readline().strip()
s2 = f.readline().strip()
f.close()
res = 0
if len(s1) != len(s2):
    res = -1
elif s1 == s2:
    res = 0
else:
    for i in range(len(s1)):
        shift = s1[len(s1) - i - 1:len(s1)] +
s1[0:len(s1) - i - 1]
        if shift == s2:
            res += 1
            break
        res += 1
    if res == len(s1):
        res = -1
w = open('output.txt', 'w')
w.write(str(res))
w.close()
```

Результат работы кода:

Тест	Результат	Время	Память
1	Accepted	0,015	382 Кб
2	Accepted	0,015	374 Кб
3	Accepted	0,046	370 Кб
4	Accepted	0,031	374 Кб
5	Accepted	0,015	370 Кб
6	Accepted	0,015	374 Кб
7	Accepted	0,015	374 Кб
8	Accepted	0,015	434 Кб
9	Accepted	0,046	922 Кб
10	Accepted	0,046	3006 Кб
11	Accepted	0,062	3146 Кб
12	Accepted	0,078	3138 Кб
13	Accepted	0,078	3138 Кб
14	Accepted	0,046	3146 Кб
15	Accepted	0,062	3134 Кб
16	Accepted	0,062	3134 Кб
17	Accepted	0,046	3142 Кб
18	Accepted	0,015	362 Кб

Вывод по задаче: реализованный алгоритм просто проверяет равенство строк, сложность O(n).

Вывод

В этой лабораторной работе я изучила алгоритмы префиксных и z-функций, а также основные алгоритмы работы со строками. Закончила делать последнюю лабораторную работу по второму семестру АСИДа!!!