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

Выполнила:

Олейник П.Д.

K3143

Проверил:

Афанасьев А.В.

Санкт-Петербург 2024 г.

Оглавление

Задачи	3
Задача № 1. Наивный поиск подстроки в строке	
Задача № 5. Префикс-функция	
Задача № 6. Декомпозиция строки	
Вывод	

Задачи

Задача № 1. Наивный поиск подстроки в строке

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.
- Пример:

input.txt	output.txt				
aba	2				
abaCaba	1 5				

Листинг кода:

```
import time
import tracemalloc
t_start = time.perf_counter()
tracemalloc.start()
def FindPatternNaive(p, t): # возвращает индексы вхождения строки р в строку
    result = []
    for i in range(len(t) - len(p) + 1):
        if AreEqual(t[i: i + len(p)], p):
            result.append(i)
    return [i + 1 for i in result] # увеличиваем каждый индекс на 1, тк
нумерация с единицы
def AreEqual(s1, s2): # проверяет, равны ли строки
    if len(s1) != len(s2):
    for i in range(len(s1)):
        if s1[i] != s2[i]:
           return False
    return True
def solve(p, t): # находит вхождения строки р в строку t
    result = FindPatternNaive(p, t)
    return len(result), result
```

```
def write_to_file():
    f = open("input1.txt")
    p = f.readline().rstrip()
    t = f.readline().rstrip()
    f.close()

    n, indexes = solve(p, t)

    file2 = open("output1.txt", "w+")
    file2.write(str(n) + "\n")
    file2.write(" ".join([str(i) for i in indexes]) + "\n")
    file2.close()

if __name__ == "__main__":
    write_to_file()

    print("Время выполнения: %s секунд " % (time.perf_counter() - t_start))
    print("Затраты памяти: %s КБ " % (tracemalloc.get_traced_memory()[0] / 2
** 10))
```

Текстовое объяснение решения.

Пусть длина строки р равна m. Проходим каждый элемент строки t (кроме последних m-1) Проверяем, равна ли подстрока с началом в текущем элементе и длиной m строке t, сравнивая их посимвольно с помощью функции AreEqual.

Результат работы кода на примерах из текста задачи:



Время выполнения: 0.0006979000172577798 секунд

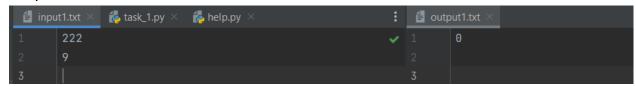
Затраты памяти: 3.6162109375 КБ

Результат работы кода на максимальных и минимальных значениях:



Время выполнения: 0.012776199990184978 секунд

Затраты памяти: 3.7724609375 КБ



Время выполнения: 0.005699100001947954 секунд

Затраты памяти: 3.2021484375 КБ

Вывод по задаче:

Для небольших длин строк поиск подстроки в строке можно осуществлять с помощью наивного алгоритма поиска.

Задача № 5. Префикс-функция

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:

ĺ		output.txt		output.txt			
ĺ	aaaAAA	012000	abacaba	0010123			

Листинг кода:

```
""Эффективный алгоритм префикс функции"""
import time
import tracemalloc
t_start = time.perf_counter()
tracemalloc.start()
def prefixFuncion(s):
    p = [0] * (len(s) + 1) # массив длин наибольших бордеров для каждого
префикса
    while i < len(s):</pre>
        if s[i] == s[j]:
            p[i+1] = j+1
        else:
                j = p[j]
            else:
                p[i+1] = 0
def solve(s): # форматирует результат prefixFuncion
    return " ".join([str(i) for i in prefixFuncion(s)[1:]])
def write_to_file():
    s = f.readline().rstrip()
    f.close()
    result = solve(s)
```

```
file2 = open("output5.txt", "w+")
file2.write(result)
file2.close()

if __name__ == "__main__":
    write_to_file()

    print("Время выполнения: %s секунд " % (time.perf_counter() - t_start))
    print("Затраты памяти: %s КБ " % (tracemalloc.get_traced_memory()[0] / 2

** 10))
```

Текстовое объяснение решения.

Префикс функция возвращает массив длин наибольших бордеров для всех префиксов данной строки. В наивном алгоритме префикс функции счетчик I по всем элементам строки (всевозможным концам префикса) и для каждого I увеличивает длину бордера (с 0) до тех пор, пока префикс и суффикс рассматриваемого суффикса совпадают. В коде используется эффективный алгоритм. Счетчик I все так же пробегает по всевозможным концам префикса. Счетчик j все так же содержит информацию о том, сколько элементов в начале совпадают с концом. Пусть в данный момент был рассмотрен элемент i-1. Если i элемент совпадает с j, то мы нашли максимальный бордер для подстроки [0; i], сохранаяем его в массив р и увеличиваем оба счетчика I и j на единицу. Если i элемент не совпал с j элементом, то нам нужно заменить j на максимальный бордер строки [0; j] и снова выполнит проверку. При этом, если в какой-то момент j оказалось равно нулю при данном i, то это значит, что значение функции для подстроки [0;i] равно нулю.

Результат работы кода на примерах из текста задачи:



Время выполнения: 0.007129299978259951 секунд

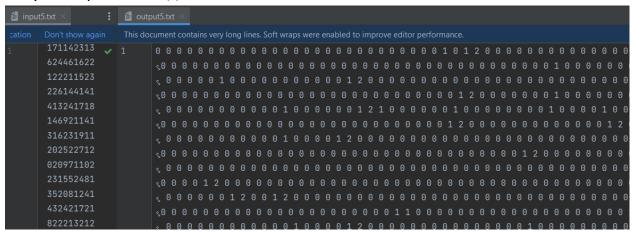
Затраты памяти: 2.1943359375 КБ



Время выполнения: 0.005577100004302338 секунд

Затраты памяти: 2.1943359375 КБ

Результат работы кода на максимальных и минимальных значениях:



Время выполнения: 0.48234170000068843 секунд

Затраты памяти: 2.1943359375 КБ

Вывод по задаче: реализован эффективный алгоритм префикс-функции.

Задача № 6. Декомпозиция строки

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:

I	input.txt	output.txt	input.txt	output.txt
ı	aaaAAA	21000	abacaba	010301

Листинг кода:

```
'""Z-бинкция"""
"""для каждого суффикса вычисляет длину максимального префикса, совпадающего
import time
import tracemalloc
t_start = time.perf_counter()
tracemalloc.start()
def z_function(s):
    values = []
    for j in range(len(s)):
        max_value_of_z_function = 0
        while j < len(s) and s[i] == s[j]:
            max_value_of_z_function += 1
        values.append(max_value_of_z_function)
    return values[1:]
def solve(s): # форматирует результат z_function
    return " ".join([str(i) for i in z_function(s)])
def write_to_file():
    f = open("input6.txt")
    s = f.readline().rstrip()
    f.close()
    result = solve(s)
    file2 = open("output6.txt", "w+")
    file2.write(result)
    file2.close()
```

```
if __name__ == "__main__":
    write_to_file()
    print("Время выполнения: %s секунд " % (time.perf_counter() - t_start))
    print("Затраты памяти: %s КБ " % (tracemalloc.get_traced_memory()[0] / 2
** 10))
```

Текстовое объяснение решения.

z_function для каждого суффикса вычисляет длину максимального префикса, совпадающего с префиксом всей строки.

Результат работы кода на примерах из текста задачи:

🛔 inpu	t6.txt ×	÷	<equation-block> outp</equation-block>	out6	.txt	×	ı	
1	aaaAAA	~	1	2	1	0	0	0

Время выполнения: 0.0008494999492540956 секунд

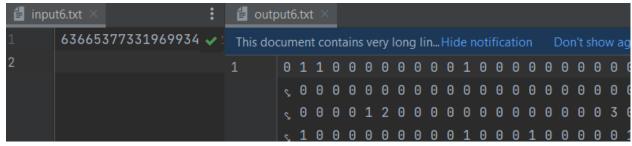
Затраты памяти: 2.5009765625 КБ



Время выполнения: 0.004012299992609769 секунд

Затраты памяти: 2.5009765625 КБ

Результат работы кода на максимальных и минимальных значениях:



Время выполнения: 0.3016310000093654 секунд

Затраты памяти: 2.5009765625 КБ

Вывод по задаче: реализован алгоритм z-функции.

Вывод

Реализованы базовые алгоритмы для работы со строками.