

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.
К3143

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Оглавление

Задачи	3
Задача № 1. Наивный поиск подстроки в строке	3
Задача № 5. Префикс-функция.....	6
Задача № 6. Декомпозиция строки	9
Вывод.....	11

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.** $1 \leq |p|, |t| \leq 10^4$.
- **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	2
abaCaba	1 5

Листинг кода:

```
"""Наивный поиск подстроки в строке"""
import time
import tracemalloc

t_start = time.perf_counter()
tracemalloc.start()

def FindPatternNaive(p, t): # возвращает индексы вхождения строки p в строку t
    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):
        return False
    for i in range(len(s1)):
        if s1[i] != s2[i]:
            return False
    return True

def solve(p, t): # находит вхождения строки p 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 p равна m . Проходим каждый элемент строки t (кроме последних $m-1$) Проверяем, равна ли подстрока с началом в текущем элементе и длиной m строке p , сравнивая их посимвольно с помощью функции `AreEqual`.

Результат работы кода на примерах из текста задачи:

input1.txt	output1.txt
1 aba	1 2
2 abaCaba	2 1 5
3	3

Время выполнения: 0.0006979000172577798 секунд

Затраты памяти: 3.6162109375 КБ

Результат работы кода на максимальных и минимальных значениях:

input1.txt	task_1.py	help.py	output1.txt
1 222	✓		1 1
2 14271932010218201451419181261437189157120154261091			2 1094
3			3

Время выполнения: 0.012776199990184978 секунд

Затраты памяти: 3.7724609375 КБ

input1.txt	task_1.py	help.py	output1.txt
1 222	✓		1 0
2 9			2
3			3

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.** $1 \leq |s| \leq 10^6$.
- **Формат вывода / выходного файла (output.txt).** Выведите значения префикс-функции для всех префиксов строки s длиной $1, 2, \dots, |s|$, 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:

```
"""Эффективный алгоритм префикс функции"""
import time
import tracemalloc

t_start = time.perf_counter()
tracemalloc.start()

def prefixFunction(s):
    p = [0] * (len(s) + 1) # массив длин наибольших бордеров для каждого префикса
    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:
                j = p[j]
            else:
                p[i+1] = 0
                i += 1
    return p

def solve(s): # форматирует результат prefixFunction
    return " ".join([str(i) for i in prefixFunction(s)[1:]])

def write_to_file():
    f = open("input5.txt")
    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 $[0; i]$, сохраняем его в массив r 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:

input5.txt	output5.txt
1	aaaAAA
✓	1
	0 1 2 0 0 0

Время выполнения: 0.007129299978259951 секунд

Затраты памяти: 2.1943359375 КБ

input5.txt	output5.txt
1	abacaba
✓	1
	0 0 1 0 1 2 3

Время выполнения: 0.005577100004302338 секунд

Затраты памяти: 2.1943359375 КБ

Результат работы кода на максимальных и минимальных значениях:

☒ input5.txt
 ☒ output5.txt

This document contains very long lines. Soft wraps were enabled to improve editor performance.

Location	Don't show again	Content
1	<input checked="" type="checkbox"/>	171142313 624461622 122211523 226144141 413241718 146921141 316231911 202522712 020971102 231552481 352081241 432421721 822213212

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.** $2 \leq |s| \leq 10^6$.
- **Формат вывода / выходного файла (output.txt).** Выведите значения Z-функции для всех индексов $1, 2, \dots, |s|$ 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:

```
"""Z-функция"""
"""для каждого суффикса вычисляет длину максимального префикса, совпадающего
с префиксом всей строки"""
import time
import tracemalloc

t_start = time.perf_counter()
tracemalloc.start()

def z_function(s):
    values = []
    for j in range(len(s)):
        i = 0
        max_value_of_z_function = 0
        while j < len(s) and s[i] == s[j]:
            max_value_of_z_function += 1
            i += 1
            j += 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()
```

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:

input6.txt			output6.txt		
1	aaaAAA	✓	1	2 1 0 0 0	

Время выполнения: 0.0008494999492540956 секунд

Затраты памяти: 2.5009765625 КБ

input6.txt x output6.txt x

1 abacaba 1 0 1 0 3 0 1

Время выполнения: 0.004012299992609769 секунд

Затраты памяти: 2.5009765625 КБ

Результат работы кода на максимальных и минимальных значениях:

[illegible]

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.