МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Поиск образца в тексте. Алгоритм Рабина-Карпа.

Студент гр. 0382 _____ Кривенцова Л.С.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Цель работы.

Изучить принцип работы алгоритма Карпа-Рабина. Реализовать с помощью него программу, производящую поиск подстроки в строке.

Задание.

Напишите программу, которая ищет все вхождения строки Pattern в строку Text, используя алгоритм Карпа-Рабина.

На вход программе подается подстрока Pattern и текст Text. Необходимо вывести индексы вхождений строки Pattern в строку Text в возрастающем порядке, используя индексацию с нуля.

Примечание: в работе запрещено использовать библиотечные реализации алгоритмов и структур.

Ограничения

$1 \le |Pattern| \le |Text| \le 5 \cdot 105.$

Суммарная длина всех вхождений образца в текста не превосходит 108. Обе строки содержат только буквы латинского алфавита.

Ограничения

$1 \le n \le 10\ 5\ ;\ 1 \le m \le 10\ 5\ ;\ 0 \le t\ i \le 10\ 9$

Пример.

Вход:

aba

abacaba

Выход:

0 4

Выполнение работы.

Программа выполнена на языке программирования Python.

Перед написанием функций, определяем глобальные константы для вычисления хэш-функции: B (константа для вычисления биномиального кольцевого хэша) и Q (количество символов в таблице ascii).

Функции:

1. *def get_hash(pattern: str) -> int*

Функция принимает на вход строку-образец, возвращает целочисленное значение – хэш подстроки.

Тело функции состоит из цикла, в котором вычисляется хэш подстроки с помощью метода Хорнера.

Для вычисления инициализируются следующие переменные:

global B, Q – сообщаем функции, что используются глобальные инициализируемые константы;

length – переменная, хранящая длину подстроки;

result – значение хэш-функции.

2. *def search_patterns_in_text(main_text: str, pattern: str)*

Функция поиска подстроки в строке. Принимает на вход исходный текст и образец, который нужно найти. Возвращает список индексов включения подстроки в строку (инициализируется как *result = []).*

Также объявляются вспомогательные переменные:

global B, Q – сообщаем функции, что используются глобальные инициализируемые константы;

pattern_len – целочисленное значение, хранящее длину образца;

main_text_len – целочисленное значение, хранящее длину исходной строки;

result = [] – список индексов включения подстроки в строку;

multiplier – максимальная степень константы B в вычислениях. Сначала инициализируем как 1, затем в цикле находим максимальное значение.

Вне циклов считаются переменные

pattern_hash = get_hash(pattern) - значение хэш-функции образца;

main_text_hash = get_hash(main_text[:pattern_len]) – значение хэш-функции первой подстроки текста, с длиной равной длине искомой подстроки.

Далее запускается цикл *for*, проходящий по исходной строке. Если хэш взятой подстроки равен хэшу паттерна, то проверяем строки на коллизию. И если подстроки совпадают, добавляем индекс начала взятой подстроки (*index_symbol)* в список результата (*result*).

Затем, после проверки на то, не вышел ли индекс за границы исходного текста, вычисляем новое значение *main_text_hash*. И если оно оказывается отрицательным, добавляем к нему константу *Q*. На этом итерация цикла завершается. Таким образом, после цикла мы имеем список со всеми индексами включения подстроки в строку и возвращаем его *(return result*).

Вне функций считываются две строки *s1* и *s2*(образец и строка для поиска, согласно условиям) и печатается отформатированный результат функции *search_patterns_in_text(s2, s1).*

Остальные функции предназначены для тестирования программы.

Тестирование.

Таблица 1. Результат тестирования.

№	Входные данные	Результат	Комментарий
1	aba abacaba	0 4	Верно
2	findf findfindfindfind	0 4 8	Верно
3	!! !!!!!!	0 1 2 3 4	Верно
4	A# A#	0	Верно
5	ABCABC abcabc		Верно

Вывод.

Был изучен принцип работы алгоритма Карпа-Рабина. Написана программа, производящая поиск подстроки в строке.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММ

Файл lb4.py

```
B = 13
Q = 256
def get_hash(pattern: str) -> int:
    global B, Q
    length = len(pattern)
    result = 0
    for i in range(length):
        result = (B * result + ord(pattern[i])) % Q
    return result

def search_patterns_in_text(main_text: str, pattern: str):
    global B, Q
    pattern_len = len(pattern)
    main_text_len = len(main_text)
    result = []
    multiplier = 1
    for i in range(1, pattern_len):
        multiplier = (multiplier * B) % Q

    pattern_hash = get_hash(pattern)
    main_text_hash = get_hash(main_text[:pattern_len])

    for index_symbol in range(main_text_len - pattern_len + 1):
        if pattern_hash == main_text_hash:
            if main_text[index_symbol: index_symbol + pattern_len]
== pattern:
                result.append(index_symbol)
        if index_symbol < main_text_len - pattern_len:
            main_text_hash = ((main_text_hash -
ord(main_text[index_symbol]) * multiplier) * B + ord(
                main_text[index_symbol + pattern_len])) % Q

            if main_text_hash < 0:
                main_text_hash += Q
    return result
```

```
    def pattern_test(test_string, pattern, answer):
        assert search_patterns_in_text(test_string, pattern) == answer,\
            "\nTest: {} in {}\nGot: {}\nExpected: {}"\
            .format(pattern,                                test_string,
search_patterns_in_text(test_string, pattern),
                    answer)
    def test_for_search_patterns():
        pattern = "aba"
        test_string = "abacaba"
        answer = [0,4]
        pattern_test(test_string, pattern, answer)

        pattern = "findf"
        test_string = "findfindfindfind"
        answer = [0,4,8]
        pattern_test(test_string, pattern, answer)

        pattern = "!!"
        test_string = "!!!!!!"
        answer = [0, 1, 2, 3, 4]
        pattern_test(test_string, pattern, answer)

        pattern = "A#"
        test_string = "A#"
        answer = [0]
        pattern_test(test_string, pattern, answer)

        pattern = "ABCABC"
        test_string = "abcabc"
        answer = []
        pattern_test(test_string, pattern, answer)

    s1 = input()
    s2 = input()
    print("  ".join(map(lambda  a:  str(a),  search_patterns_in_text(s2,
s1))))
    test_for_search_patterns()

    print("All tests were successfully passed")
```