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. 1303 _____ Беззубов Д.В.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2022

Цель работы.

Изучение хеширования данных, реализация алгоритма Рабина-Карпа, основанного на хешировании, для эффективного поиска заданной 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.

Выполнение работы.

Для решения данной задачи были реализованы следующие функции*:*

- *hashing(str, k)* – принимая на вход список множителей и строку, возвращает хеш-значение
- *substringRK(text, pattern, coefs)* – функция, реализующая алгоритм Карпа-Рабина, а именно ищет все вхождения подстроки *pattern* в строку text.

В функции *main()* считываются строка-шаблон и строка, в которой осуществляется поиск. Число X положим равным мощности алфавита введенной строки. В *main()* производится вычисление списка множителей вида {x^i | i = 0,1..m-1}.

В функции *hashing(str, k)* производится вычисление хеш-функции следующего вида:

$$H = (c_1 \times b^{m-1} + c_2 \times b^{m-2} \ldots + c_m \times b^0) \bmod Q$$

Рисунок 1 – Формула хеш-функции

Где $c_1, c_2 \ldots c_m$ – символы в строке, $b^{m-1}, b^{m-2} \ldots b^0$ – элементы списка множителей, полученного в *main()*.

В функции *substringRK(text, pattern, coefs)* сначала копируется подстрока из *text* длины подстроки *pattern*. Для *pattern* и *substr* вычисляется хеш. Затем в цикле

сравниваются полученные значения, в случае совпадения – в список *res[]* сохраняется индекс вхождения подстроки. Затем, к *substr* добавляется очередной символ из *text* и хеш пересчитывается. Можно считать, что в *substringRK(text, pattern, coefs)* используется скользящая хеш-функция, т.к. значение её вычисляется по формуле:

$$H = \left((H_p - C_p \times b^{m-1}\right) \times b + C_n) \bmod Q$$

Где *Hp* – предыдущее значение хеша, *Cp* – символ, который необходимо удалить из подстроки, *Cn* – символ, который добавили в строку.

Данная реализация позволяет избежать затрат ресурсов и времени на взятие подстрок на каждом шаге работы данного алгоритма.

Тестирование программы.

Тестирование функции производится с помощью unit-тестов, описанных в файле *test.py.* Данные тесты покрывают следующие случаи:

- На вход подается строка из одинаковых символов и шаблон из тех же символов, но меньшей длины
- На вход подается строка и шаблон, входящий в подстроку несколько раз.
- На вход подается строка и шаблон, не входящий в подстроку.
- Строка и шаблон совпадают

Вывод.

Была освоена работа с хеш-функциями. Реализована программа, основанная на алгоритме Рабина-Карпа, находящая вхождения подстроки в строку.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: substringRK.py

```
def hashing(str, k):
    res = 0
    for i in range(len(str)):
        res+=ord(str[i])*k[len(str) - i - 1]
    return res % 1000000007

def substringRK(text, pattern, coefs):
    p = 1000000007
    res = []
    pat_len = len(pattern)
    substr = text[:pat_len]

    patt_hash = hashing(pattern, coefs)
    substr_hash = hashing(substr, coefs)

    for i in range(pat_len, len(text)+1):
        if patt_hash == substr_hash:
            res.append(i - pat_len)
        if i < len(text):
            substr += text[i]
            substr_hash = ((substr_hash - ord(substr[i -
pat_len])*coefs[pat_len-1])*coefs[1] + ord(text[i])) % p
    return res

def main():
    pattern = input()
    str = input()
    m = len(set(str))
    coefs = [m**i for i in range(len(pattern))]
    res = substringRK(str, pattern, coefs)
    print(*res)

if __name__ == '__main__':
    main()
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД UNIT-ТЕСТОВ

Название файла: test.py

```
import pytest
from substringRK import *

def test_screaming():
    alph = len(set('a a a a a a'))
    coefs = [alph ** i for i in range(len('aaa'))]
    res = substringRK('aaaaaa', 'aaa', coefs)
    assert res == [0, 1, 2, 3]

def test_common():
    alph = len(set('abacaba'))
```

```
    coefs = [alph ** i for i in range(len('aba'))]
    res = substringRK('abacaba', 'aba', coefs)
    assert res == [0, 4]

def test_empty():
    alph = len(set('ᵃbacaba'))
    coefs = [alph ** i for i in range(len('aaa'))]
    res = substringRK('abacaba', 'aaa', coefs)
    assert res == []

def test_equal():
    alph = len(set('ᵃbacaba'))
    coefs = [alph ** i for i in range(len('ᵃbacaba'))]
    res = substringRK('ᵃbacaba', 'ᵃbacaba', coefs)
    assert res == [0]
```