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. 0304 _____ Мажуга Д. Р.

Преподаватель _____ Берленко Т. А.

Санкт-Петербург
2021

Цель работы.

Изучить работу с хеш-функцией и алгоритм Рабина-Карпа для нахождения вхождений подстроки в строку.

Задание.

Поиск образца в тексте. Алгоритм Рабина-Карпа. Python

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.

Пример:

Вход: aba abacaba

Выход: 0 4

Основные теоретические положения.

Алгоритм Рабина-Карпа — алгоритм поиска строки, который ищет шаблон (подстроку) в тексте, используя хеширование. Сложность алгоритма в худшем случае O(nm), в лучшем O(n+m), где n — длина текста, m — длина строки.

Выполнение работы.

Был реализован класс *RabinKarpAlghoritm* в котором была реализована хеш функциия *hashing(string)* — в данном методе мы возвращаем сумму всех символов строки преведённых в *ascii*.

Также был реализован метод *matcher(pattern, string)* который реализует поиск всех вхождений pattern в строку используя алгоритм Рабина-Карпа. Сначала в пременные *h* и *len_pat* мы сохранем хеш строки *pattern* и длину строки *pattern*, после чего в переменную *sub_h* мы хешируем строку до длины строки *pattern*, далее проверяем совпадает ли хеш *pattern* с хешом нашей

подстрокой(*sub_h*) и совпадает ли подстрока с *pattern*, если все условия выполнены мы добавляем индекс первого вхождения в массив *indexes*. После чего мы в цикле на каждой итерации высчитываем новый хеш новой подстроки, после чего проверяем не совпадает ли хеш нашего паттерна с построкой, а также проверяем совпадают ли символы подстроки и паттерна, если условия выполнены, то добавляем индекс вхождения в массив *indexes*.

Также был реализован класс *Interface* в конструкторе которого мы создаем экземпляр класса *RabinKarpAlghoritm*, а в статическом методе run мы считываем паттерн и строку и вызываем метод matcher класса *RabinKarpAlghoritm*, после чего выводим результат на экран.

Тестирование.

Были написанны тесты, необходимые для проверки корректности результатов программы.Результаты тестирования представлены в табл. 1.

№	Входные данные	Выходные данные	Комментарий
1	aba abacaba	0 4	OK
2	aaaaa aaaaaaaaaaa	0 1 2 3 4 5 6 7	OK
3	Hello Hello my friend. Do you know how print Hellow wolrd on c++	0 39	OK

Таблица 1 — Результаты тестирования

Вывод.

Был изучен алгоритм Рабина-Карпа и основы работы с ним. Была написана программа, принимающая на вход подстроку и основной текст, находящая при помощи алгоритма Рабина-Карпа все вхождения подстроки в текст и выводящая в терминал индекс первого символа каждого вхождения.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Alghoritm.py

```
class RabinKarpAlghoritm:
    @staticmethod
    def hashing(string):
        return sum(ord(cymb) for cymb in
string)


    def matcher(self, pattern, string):
        h, len_pat = self.hashing(pattern),
len(pattern)
        sub_h =
self.hashing(string[:len_pat])
        indexes = [0] if h == sub_h and
string[:len_pat] == pattern else []
        for i in range(1, len(string) -
len_pat + 1):
            sub_h = sub_h - ord(string[i-1])
+ ord(string[i+len_pat-1])
            if h == sub_h and
string[i:i+len_pat] == pattern:
                indexes.append(i)
        return indexes



class Inerface:
    def __init__(self):
        self.alghoritm =
RabinKarpAlghoritm


    @staticmethod
    def run():
        pattern, string = input(), input()
        answer =
RabinKarpAlghoritm().matcher(pattern,
string)
        print(*answer)
```

Название файла: main.py

```
from Alghoritm import Inerface



if __name__ == '__main__':
    match = Inerface()
    match.run()
```

Название файла: tests.py

```
from Alghoritm import
RabinKarpAlghoritm


class TestCases:
    @staticmethod
    def frst_test():
        answer =
RabinKarpAlghoritm().matcher('aba',
'abacaba')
        guess_value = [0, 4]
        assert answer == guess_value
        print('test first: ok')


    @staticmethod
    def scnd_test():
        answer =
RabinKarpAlghoritm().matcher('aaaaa',
'aaaaaaaaaaa')
        guess_value = [0, 1, 2, 3, 4, 5, 6,
7]
        assert answer == guess_value
        print('test second: ok')


    @staticmethod
```

```
    def thrd_test():
        answer =
RabinKarpAlghoritm().matcher('aba',
'qwertyuiopmnbbvccxz')
        guess_value = []
        assert answer == guess_value
        print('test third: ok')


    @staticmethod
    def four_test():
        answer =
RabinKarpAlghoritm().matcher('Hello',
'Hello my friend. Do you know how print
Hellow wolrd on c++')
        guess_value = [0, 39]
        assert answer == guess_value
        print('test four: ok')


if __name__ == '__main__':
    case = TestCases
    case.frst_test()
    case.scnd_test()
    case.thrd_test()
    case.four_test()
```