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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Поиск образца в тексте. Алгоритм Рабина-Карпа

Студентка гр.1381 _____ Рымарь М.И.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2022

Цель работы.

Изучить хэш-функции и хэш-таблицы. Реализовать решение задачи лабораторной работы с использованием хэш-функции.

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

$1 \le |\text{Pattern}| \le |\text{Text}| \le 5 \cdot 105$.

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

Пример.

Вход:

aba

abacaba

Выход:

0 4

Подсказки:

1. Будьте осторожны с операцией взятия подстроки — она может оказаться дорогой по времени и по памяти.

2. Храните степени x ** p в списке - тогда вам не придется вычислять их каждый раз заново.

Выполнение работы.

В ходе выполнения лабораторной работы помимо основной функции, реализующей ввод и вывод, была написана функция, выполняющая алгоритм Рабина-Карпа. Подфункция и текст считываются посимвольно в два массива, каждый символ меняется на его ASCII код и вычитается ASCII код заглавной ‘A’.

В основной функции хеширования берутся два простых числа, которые в дальнейшем будут использоваться для уменьшения вероятности переполнения. Создаётся массив для ответа и массив степеней для применения в алгоритме. Чтобы предотвратить коллизию, делаем проверку первой строчкой в цикле. Отдельно рассчитываем хэш подстроки. Чтобы на каждом шаге не высчитывать последние мономы (каждое число, умноженное на простое число в старшей степени), создаётся массив последних мономов. Далее высчитываем хэш первой подстроки. Далее в цикле проверяем совпадение хэша и подстроки. Если совпадает, то добавляем в массив ответов. Далее делаем пересчёт хэша и делаем проверку для случая коллизии. Функция возвращает массив индексов подстрок.

Тестирование.

Для тестирования используется фреймворк unittest. Тесты рассматривают несколько случаев с разными значениями строк и подстрок, в том числе граничные значения. Результаты тестирования представлены на рисунке 1. Файл с тестированием *test.py* представлен в приложении А.

```
============================= test session starts ==============================
collecting ... collected 5 items

test.py::TestHash::test1 PASSED                                          [ 20%]
test.py::TestHash::test2 PASSED                                          [ 40%]
test.py::TestHash::test3 PASSED                                          [ 60%]
test.py::TestHash::test4 PASSED                                          [ 80%]
test.py::TestHash::test5 PASSED                                          [100%]

============================== 5 passed in 0.02s ===============================

Process finished with exit code 0
```

Рисунок 1 – Результаты тестирования

Выводы.

В ходе выполнения лабораторной работы была изучена такая структура данных, как хэш-функция. Был применен алгоритм Рабина-Карпа для реализации решения задания.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММ

Название файла: *main.py*

```
def main():
    pattern = [ord(s) - 65 for s in input()]
    text = [ord(s) - 65 for s in input()]
    print(*hash(pattern, text))

def hash(pattern, text):
    x, p = 59, 67
    answer = []
    x_pows = [1]
    for i in range(1, len(pattern)):
        x_pows.append(x_pows[-1] * x % p)

    pattern_hash = sum([pattern[i] * x_pows[len(pattern) - i - 1] for i in
range(len(pattern))]) % p

    last_monoms = [text[i] * x_pows[-1] % p for i in range(len(text) -
len(pattern) + 1)]

    cur_hash = (last_monoms[0] + sum(
        [text[i] * x_pows[len(pattern) - i - 1] for i in range(1,
len(pattern))])) % p

    for i in range(len(text) - len(pattern)):

        if pattern_hash == cur_hash and pattern == text[i:(i +
len(pattern))]:
            answer.append(i)

        cur_hash = ((cur_hash - last_monoms[i]) * x + text[
            i + len(pattern)]) % p

    if pattern_hash == cur_hash and pattern == text[(
            len(text) - len(pattern)):]:
        answer.append(len(text) - len(pattern))
    return answer

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

Название файла: *test.py*

```
import unittest
from main import hash

class TestHash(unittest.TestCase):

    def test1(self):
        pattern = [ord(s) - 65 for s in "aba"]
        text = [ord(s) - 65 for s in "abacaba"]
        self.assertEqual(hash(pattern, text), [0, 4])

    def test2(self):
        pattern = [ord(s) - 65 for s in "Test"]
        text = [ord(s) - 65 for s in "testTesttesT"]
        self.assertEqual(hash(pattern, text), [4])

    def test3(self):
        pattern = [ord(s) - 65 for s in "aaaaa"]
        text = [ord(s) - 65 for s in "baaaaaaa"]
        self.assertEqual(hash(pattern, text), [1, 2, 3])

    def test4(self):
        pattern = [ord(s) - 65 for s in "mnk"]
        text = [ord(s) - 65 for s in "12345"]
        self.assertEqual(hash(pattern, text), [])

    def test5(self):
        pattern = [ord(s) - 65 for s in "mnk"]
        text = [ord(s) - 65 for s in "mnk"]
        self.assertEqual(hash(pattern, text), [0])
```