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.

Студентка гр. 2384 Валеева А.А.

Преподаватель Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Изучить структуру хэш-таблиц и принцип работы хэширования. Затем реализовать алгоритм Рабина-Карпа для поиска подстроки в тексте. Исследовать асимптотику на различных размерах данных.

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

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

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

Пример.

Вход:

aba

abacaba

Выход:

0 4

Теоретическая справка

Алгоритм Рабина — Карпа — это алгоритм поиска строки, который ищет шаблон, то есть подстроку, в тексте, используя хеширование. Он был разработан в 1987 году Михаэлем Рабином и Ричардом Карпом.

Алгоритм редко используется для поиска одиночного шаблона, но имеет значительную теоретическую важность и очень эффективен в поиске совпадений множественных шаблонов одинаковой длины. Для текста длины n и шаблона длины m его среднее и лучшее время исполнения равно O(n) при правильном выборе хеш-функции, но в худшем случае он имеет эффективность O(nm), что является одной из причин того, почему он не слишком широко используется. Для приложений, в которых допустимы ложные срабатывания при поиске, то есть, когда некоторые из найденных вхождений шаблона на самом деле могут не соответствовать шаблону, алгоритм Рабина — Карпа работает за гарантированное время O(n) и при подходящем выборе рандомизированной хеш-функции вероятность ошибки можно сделать очень малой. Также алгоритм имеет уникальную особенность находить любую из заданных k строк одинаковой длины в среднем (при правильном выборе хеш-функции) за время O(n) независимо от размера k.

Выполнение работы

Константы: за мощность алфавита (*d*) взято значение 256, равное размеру 8-битной таблице ASCII. За простое число *q* взято значение 101.

В реализованном алгоритме используется данный метод хэширования: для каждого символа s, находящегося по индексу i в подстроке, значения хэша высчитывается по формуле *(d * hash_pattern + ASCII(s)) %q.*

Далее выполняется проход по тексту, если хэш-значение подстроки и части текста совпали, то проверяем их значения посимвольно, чтобы исключить возможность коллизии, для проверки вводим переменную проверки *flag.* Если строки совпали, добавляем индекс начала данного кусочка текста в массив *result.*

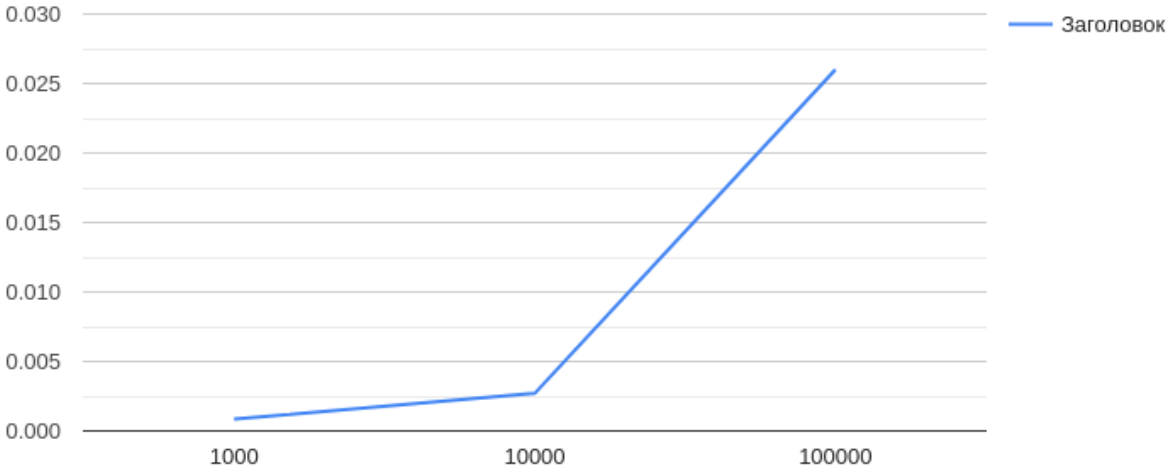
После этого нужно сделать сдвиг на один символ, чтобы не считать заново все хэш-значение новой подстроки текста, берем хэш-значение предыдущей подстроки, удаляем ASCII код первого символа, который отпадет при сдвиге вправо, и прибавляем ASCII код следующего символа.

Было проведено тестирование скорости выполнения работы на разных размерах данных. Была использована подстрока размером 20 символов. Результаты тестирования см. в таблице 1.

Набор данных	1 000	10 000	100 000
Алгоритм Рабина-Карпа	0.00086 секунд	0.0027 секунд	0.026 секунд

Таблица 1 – Результаты замеров

Разработанный программный код см. в приложении А.



ис. 1 – Время работы алгоритма Рабина-Карпа на разных наборах данных

Выводы

По результатам лабораторной работы был реализован алгоритм Рабина-Карпа на языке Python, а также было разобрано и использовано его тестирование с помощью библиотеки Pytest, были проведены замеры времени работы алгоритма на различных размерах входных данных.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: algoritm.py

```
def algoritm_rabin_karp(pat, txt, q): #q - простое число
    d = 256 # мощность алфавита
    pattern_len = len(pat)
    text_len = len(txt)
    result = []
    hash_pattern = 0
    hash_text = 0
    h = d**(pattern_len - 1) % q

    if (pattern_len <= text_len):
    # вычисляем хеш-знчение для искомой строки и первого окна
текста
        for i in range(pattern_len):
            hash_pattern = (d * hash_pattern + ord(pat[i])) % q
            hash_text = (d*hash_text + ord(txt[i])) % q
    else:
        print("Введен слишком большой паттер")

    for i in range(text_len - pattern_len + 1):
        flag = 1
        if hash_text == hash_pattern: #проверим - коллизия или
правда искомая подстрока
            for j in range(pattern_len): # проходим посимвольно по
паттерну
                if txt[i+j] != pat[j]:
                    flag = 0
                    break # коллизия
            if flag == 1:
                result.append(i)

        # вычисляем хеш-значение для следующего окна текста
        if i < text_len - pattern_len: #если оставшийся кусок
текста меньше длины паттерна
            # для сдвига убираем уже ненужный символ и добавляем
следующий символ строки текста
            hash_text = (d * (hash_text - ord(txt[i]) * h) +
ord(txt[i + pattern_len])) % q

            # преобразуем отрицательные значения t в положительные
при необходимости
            if hash_text < 0:
                hash_text += q

    return result #массив с индексами вхождений подстроки в текст
```

Название файла: main.py

```
from algoritm import algoritm_rabin_karp
import time

if __name__ == '__main__':
    my_pat = input()
    my_text = input()
    start_time = time.time()  # время начала выполнения
    answer = algoritm_rabin_karp(my_pat, my_text,  101)
    print(*answer)
```

Название файла: tests.py

```
from algoritm import algoritm_rabin_karp

def test_moevm():
    pat = "aba"
    txt = "abacaba"
    q = 101
    my_answer = algoritm_rabin_karp(pat, txt, q)
    answer = [0, 4]
    assert my_answer == answer

def test_empty():
    pat = "aba"
    txt = "abccabd"
    q = 101
    my_answer = algoritm_rabin_karp(pat, txt, q)
    answer = []
    assert my_answer == answer

def test_equal():
    pat = "aba"
    txt = "aba"
    q = 101
    my_answer = algoritm_rabin_karp(pat, txt, q)
    answer = [0]
    assert my_answer == answer

def test_large_values():
    pat = "cbhadgfvcagvc"
    txt =
"cbhadgfvcagvcvbshcbdchvbhdsbjkcvhbwdsjkcbcbhadgfvcagvcwdjscvhbdcb
hadgfvcagvcsjbvsnc"
    q = 101
    my_answer = algoritm_rabin_karp(pat, txt, q)
    answer = [0, 41, 63]
    assert my_answer == answer

def test_incorrect_len():
    pat = "cbhadgfvcagvc"
```

```
txt = "a"
q = 101
my_answer = algoritm_rabin_karp(pat, txt, q)
answer = []
assert my_answer == answer
```