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. 1304 _____ Шаврин А.П.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2022

Цель работы.

Изучит алгоритм Рабина-Карпа поиска образца в тексте и реализовать данный алгоритм.

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

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

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

Пример.

Вход:

aba

abacaba

Выход:

0 4

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.

Первой строкой добавьте #python или #c++, чтобы проверяющая система знала, каким языком вы пользуетесь.

Выполнение работы.

1. Сначала был реализован класс *RabinKarp*

В методе *__init__*, принимающем искомую подстроку и текст, происходит инициализация всех необходимых констант:

- *q = 5* – любое простое число для функции хэширования
- *x = 3* – любое число в диапазоне от 0 до *q – 1* для функции хэширования
- *pattern* – искомая подстрока
- *text* – текст, в котором ищется подстрока
- *len_pattern* – длина искомой подстроки
- *len_text* – длина текста
- *x_arr* – массив, в котором хранятся значения x в степенях от 0 до *len_pattern*
- *hpattern* – хэш искомой подстроки
- *answer* – массив, хранящий индексы вхождения подстроки в тексте

Потом был написан метод *__createXArr*, заполняющий массив *x_arr* необходимыми значениями.

Затем был написан метод *__hashing*, получающий на вход строку и возвращающий ее хэш значение.

После был написан метод *search*, в котором происходит поиск подстроки в тексте. Сначала в переменную *htext* записывается хэш текста с 0 по *len_pattern* символ. После в цикле происходит проверка совпадения данного хэша с хэшем искомой подстроки, если это так, то проверяется совпадение искомой подстроки с рассматриваемой. Если все условия выполнились, в *answer* добавляется индекс начала подстроки. Потом идет проверка, можем ли мы смести хэш (перейти к следующей подстроке), если это так, то делаем это и цикл начинается заново.

В конце был реализован метод *getAnswer*, возвращающий список индексов, вхождений искомой подстроки в тексте.

2. Затем была реализована основная логика работы программы.

Сначала было реализовано считывание параметров *pattern* и *text*, которые затем передаются в функцию *main*

В функции *main* создается объекта класса *RabinKarp* с передачей в конструктор искомой подстроки и самого текста.

Затем вызывается метод *search*, через объект класса *RabinKarp*.

А в конце возвращается значение, которое вернет метод класса *getAnswer*, после поиска всех вхождений подстроки.

Значение, которое вернула функция *main* выводится.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

Изучен алгоритм Рабина-Карпа поиска образца в тексте и реализован данный алгоритм.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: RabinKarp.py

```
#python

class RabinKarp():
    # инициализация необходимых констант
    def __init__(self, pattern, text):
        self.q = 5      # q - любое простое число
        self.x = 3      # x - любое число от 0 до q - 1

        self.pattern = pattern
        self.text = text

        self.len_pattern = len(pattern)
        self.len_text = len(text)

        # массив степеней x от x ** 0 до x ** len_pattern
        self.x_arr = [0] * self.len_pattern
        self.__createXArr()

        # сохранение хэша искомой подстроки
        self.hpattern = self.__hashing(self.pattern)

        self.answer = []

    # создает массив степеней x от x ** 0 до x ** len_pattern
    def __createXArr(self):
        for i in range(self.len_pattern):
            self.x_arr[i] = self.x ** i

    # хэширование строки из алгоритма Рабина-Карпа
    def __hashing(self, string):
        res = 0
        m = len(string)
        for i in range(m):
```

```
                res += (ord(string[i]) * self.x_arr[m - i - 1]) %
self.q

            return res % self.q

        # поиск вхождения подстроки в тексте
        def search(self):
            htext = self.__hashing(self.text[:self.len_pattern])

            if self.len_pattern != 0 and self.len_text != 0:
                for i in range(self.len_text - self.len_pattern + 1):
#  self.len_text  -  self.len_pattern  +  1,  чтобы  проверить  последнюю
подпоследовательность в тексте
                    if htext == self.hpattern:
                        if   self.pattern   ==   self.text[i   :   i   +
self.len_pattern]:
                            self.answer.append(i)

                    # для того, что бы не было ошибки при проверке
последней подпоследовательности в тексте
                    if (i + self.len_pattern) < self.len_text:
                        first_symb_code = ord(self.text[i])
                        next_symb_code      =      ord(self.text[i      +
self.len_pattern])

                        htext    =    ((htext    -    first_symb_code    *
self.x_arr[-1]) * self.x_arr[1] + next_symb_code) % self.q

        # возвращает результат
        def getAnswer(self):
            return self.answer


    def main(pattern, text):
        rabin_karp = RabinKarp(pattern, text)
        rabin_karp.search()
        return rabin_karp.getAnswer()
```

```
if __name__ == "__main__":
    pattern = input()
    text = input()
print(*main(pattern, text))
```

Название файла: pytests.py

```
from RabinKarp import main

def test_mv():
    pattern = "aba"
    text = "abacaba"
    assert main(pattern, text) == [0, 4]

def test_null_pattern():
    pattern = ""
    text = "sdfqwesd"
    assert main(pattern, text) == []

def test_null_text():
    pattern = "asdf"
    text = ""
    assert main(pattern, text) == []

def test_pattern_at_the_beginning_of_the_text():
    pattern = "asdasd"
    text = "asdasdxcvbnmkiuyt"
    assert main(pattern, text) == [0]

def test_pattern_at_the_end_of_the_text():
    pattern = "asdasd"
    text = "xcvbnmkiuytasdasd"
    assert main(pattern, text) == [11]

def test_text_without_pattern():
    pattern = "fgvjbk"
    text = "ffghjrtyui vjhgf"
    assert main(pattern, text) == []

def test_pattern_with_spaces_and_other_symbols():
```

```
        pattern = "Он (Algorithm)~был| разработан в [$1987] {году'!"
        text = "Алгоритм Рабина. Он (Algorithm)~был| разработан в [$1987] {году'! Майклом Рабином и Ричардом Карпом.[1]"
    assert main(pattern, text) == [17]
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные
1.	aba abacaba	0 4
2.	 sdfqwesd	
3.	asdf	
4.	asdasd asdasdxcvbnmkiuy	0
5.	asdasd xcvbnmkiuytasdasd	11
6.	fgvjbk ffghjrtyui vjhgf	
7.	Он (Algorithm)~был| разработан в [$1987] {году'! Алгоритм Рабина. Он (Algorithm)~был| разработан в [$1987] {году'! Майклом Рабином и Ричардом Карпом.[1]	17