

Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Курский государственный университет»

Кафедра программного
обеспечения и администрирования
информационных систем
Направление подготовки
математическое обеспечение и
администрирование
информационных систем
Форма обучения очная

Отчет

по лабораторной работе №2

«Сравнительный анализ методов поиска подстроки в строке»

Выполнил:

студент группы 213

Тихонов Е.Е.

Проверил:

профессор кафедры ПОиАИС

Кудинов В.А.

Курск, 2021

Цель работы: изучение методов поиска подстроки в строке и приобретение навыков в проведении сравнительного анализа различных методов.

Задание

1. Изучить временные характеристики алгоритмов.
2. Изучить методы поиска подстроки в строке.
3. Программно реализовать 2 метода: алгоритм Shift-Or, алгоритм Бойера-Мура.
4. Разработать и программно реализовать средство для проведения экспериментов по определению временных характеристик алгоритмов.
5. Определить порядок функций временной сложности алгоритмов.

Алгоритмы решения задач

Для алгоритма Shift-Or сначала строится матрица состоящая из единиц размера $n \cdot (m+1)$, где n - длина искомого паттерна, m - длина строки. Вводится операция сдвига столбца, при котором последний элемент теряется, а в начало добавляется 0. Вводится операция получения битной маски паттерна по символу K . Данная операция возвращает вектор длины n (как длина паттерна), состоящий из нулей и единиц. Нули стоят на тех местах маски, где есть вхождение символа K в паттерн. Далее в цикле для всех столбцов, начиная с нулевого столбца матрицы, применяем к столбцу операцию сдвига, а к символу (соответствующему номеру сдвинутого столбца) строки, в которой ищем паттерн, применяем операцию получения битной маски. Далее выполняем побитовое ИЛИ для столбца (который мы сдвигали в первом шаге) и битной маски (которую мы получали по символу строки во втором шаге). Полученный результат операции записываем в следующий столбец за сдвинутым. Когда окажется, что последний элемент столбца 0, то мы нашли полное вхождение паттерна в строку. Не обязательно хранить всю матрицу в памяти, достаточно хранить предыдущий и текущий столбец.

Алгоритм Бойера-Мура включает в себя предобработку искомого паттерна. Мы составляем словарь-алфавит из символов паттерна, где каждому ключу (символу) присваиваем значение расстояния от этого символа в паттерне до конца паттерна. Схема поиска паттерна в строке следующая. Мы двигаемся по строке слева направо, сравниваем символы паттерна же мы справа налево. То есть мы начинаем поиск не с нулевого символа строки а с $n-1$, где n - длина паттерна. Также мы делаем прыжки на расстояние n если символ строки не входит в наш словарь-алфавит. Если при сравнении паттерна и строки символы не совпали, но при этом

сравниваемый символ в строке содержится в нашем словаре-алфавите то мы делаем прыжок на значение лежащее по данному ключу(символу).

Сравнительный анализ алгоритмов

Временные затраты алгоритмов для разных строк показаны в таблице 1.

Таблица 1

Алгоритм	Количество символов в строке			
	60	120	240	480
Shift-Or	148348 ns	296249 ns	555988 ns	1212235 ns
Бойера-Мура	18000 ns	29068 ns	52670 ns	127286 ns

Общая оценка сложности алгоритмов показана в таблице 2.

Таблица 2

Сортировка	Сложность
Shift-Or	$n * m$ (где n - длина паттерна, m - длина строки)
Бойера-Мура	$n + m$ (где n - длина паттерна, m - длина строки)

Текст программы

Текст файла shift_or.py

```
def shift_or(str, substr): #shift_and
    n = len(substr)
    m = len(str)
    old_mask = [1] * n #[0] * n
    enter_inds = []
    for i in range(m):
        bit_shift(old_mask)
        mask = bit_mask(str[i], substr)
        old_mask = bit_or(old_mask, mask) #bit_and(old_mask, mask)
```

```

        if old_mask[len(old_mask)-1] == 0: # == 1
            enter_inds.append(i-n+1)
    return enter_inds

def bit_mask(symb, substr):
    n = len(substr)
    mask = [1] * n #[0] * n
    for i in range(n):
        if substr[i] == symb:
            mask[i] = 0 #mask[i] = 1
    return mask

def bit_shift(old_mask):
    old_mask.pop()
    old_mask.insert(0, 0) #old_mask.insert(0, 1)

def bit_or(mask1, mask2):
    return [a|b for a,b in zip(mask1,mask2)]
#for shift-and
def bit_and(mask1, mask2):
    return [a*b for a,b in zip(mask1,mask2)]

```

Текст файла boyer_mur.py

```

def pred_compile_dict(substr):
    dict = {} #char - position
    substr_len = len(substr)
    for i in range(substr_len-1):
        #сколько символов (считая справа подстроки) до этого символа
        dict[substr[i]] = substr_len - i - 1
    dict[substr[substr_len-1]] = 1
    return dict

def boyer_mur(str, substr):
    enter_inds = []
    dict = pred_compile_dict(substr)
    # k-индекс для прохода по строке (справа налево)
    # j-индекс для прохода по подстроке (справа налево)
    # i-индекс указывает на место откуда начался проход по строке (слева направо)
    substr_last_ind = i = j = k = len(substr) - 1
    while i < len(str):

```

```

        #символы совпали - движемся по подстроке дальше (от конца к
началу)

        #и сравниваем остальные символы
        if str[k] == substr[j]:
            k-=1
            j-=1

        #символы не совпали - перемещаемся на следующее
        #вхождение последнего символа подстроки в строку
        else:
            i += dict.get(str[i], substr_last_ind+1)
            k = i
            j = substr_last_ind

        if j < 0: #нашли
            enter_inds.append(k+1)
            i += dict.get(str[i], substr_last_ind+1)
            k = i
            j = substr_last_ind

    return enter_inds

```

Текст файла main.py

```

from shift_or import shift_or
from boyer_mur import boyer_mur

import time

str1 = "jhghjhjh abc hfbb bdfdjhghjhjyjhvsdrvjhghbbdjhge tthghghgh"
substr = "jhghh"

print(len(str1))

print("shift-or")
t_start = time.monotonic_ns()
inds = shift_or(str1, substr)
t_stop = time.monotonic_ns()
#test
for i in inds:
    print(str1[i:i+len(substr)])
print(str(t_stop - t_start) + " ns")

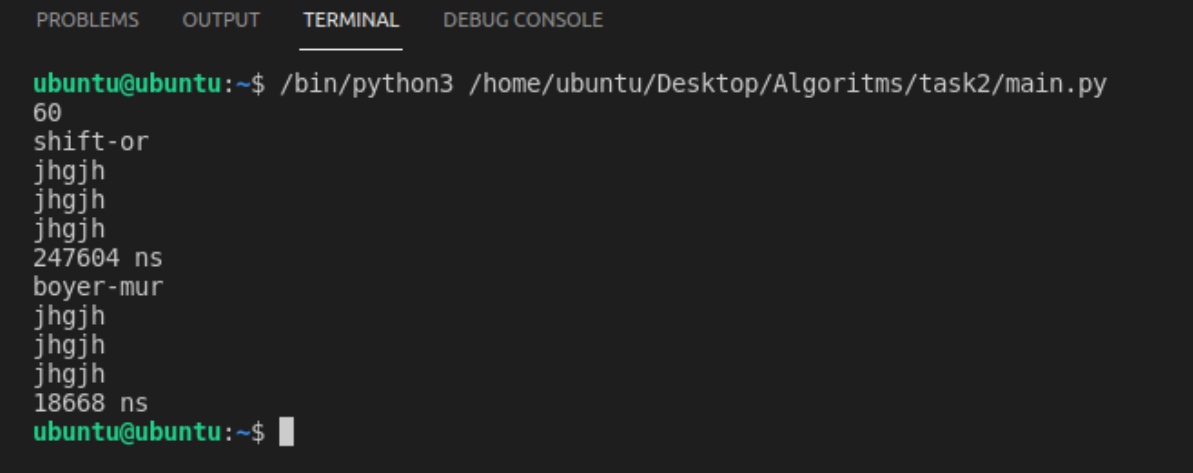
print("boyer-mur")

```

```
t_start = time.monotonic_ns()
inds = boyer_mur(str1, substr)
t_stop = time.monotonic_ns()
#test
for i in inds:
    print(str1[i:i+len(substr)])
print(str(t_stop - t_start) + " ns")
```

Тест программы

Вывод программы представлен на рисунке 1.



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

ubuntu@ubuntu:~$ /bin/python3 /home/ubuntu/Desktop/Algoritms/task2/main.py
60
shift-or
jhgjh
jhgjh
jhgjh
247604 ns
boyer-mur
jhgjh
jhgjh
jhgjh
18668 ns
ubuntu@ubuntu:~$
```

Рисунок 1 - Вывод программы.