|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Лабораторная работа №7*

*По предмету: «Анализ алгоритмов»*

**Поиск подстроки в строке**

Студент: Гасанзаде М.А.,

Группа: ИУ7-56Б

Москва, 2019 г.

Оглавление

[Введение 3](#_Toc27835170)

[1. Аналитическая часть 4](#_Toc27835171)

[1.1. Описание алгоритмов 4](#_Toc27835172)

[1.1.2 Алгоритм Бойера-Мура 5](#_Toc27835173)

[1.1.3 Алгоритм Кнута-Морриса-Пратта 5](#_Toc27835174)

[1.2. Применение алгоритмов 6](#_Toc27835175)

[2. Конструкторская часть 7](#_Toc27835176)

[2.1 Требования к программе 7](#_Toc27835177)

[2.2 Пример работы алгоритмов 7](#_Toc27835178)

[2.3 Вывод 8](#_Toc27835179)

[3. Технологическая часть 9](#_Toc27835180)

[3.1 Требования к программному обеспечению 9](#_Toc27835181)

[3.2 Средства реализации 9](#_Toc27835182)

[3.3 Листинг кода 9](#_Toc27835183)

[3.4 Описание тестирования 11](#_Toc27835184)

[3.5 Вывод 11](#_Toc27835185)

[4. Экспериментальная часть 12](#_Toc27835186)

[4.1. Примеры работы 12](#_Toc27835187)

[4.2. Результаты тестирования 12](#_Toc27835188)

[4.3. Постановка эксперимента по замеру времени 13](#_Toc27835189)

[4.4. Сравнительный анализ на материале экспериментальных данных 13](#_Toc27835190)

[4.5 Вывод 15](#_Toc27835191)

[Заключение 16](#_Toc27835192)

[Список литературы 17](#_Toc27835193)

# Введение

 Одна из простейших задач [поиска информации](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B8%D1%81%D0%BA_%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%B8). Применяется в виде встроенной функции в [текстовых редакторах](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D1%80%D0%B5%D0%B4%D0%B0%D0%BA%D1%82%D0%BE%D1%80), [СУБД](https://ru.wikipedia.org/wiki/%D0%A1%D0%A3%D0%91%D0%94), [поисковых машинах](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B8%D1%81%D0%BA%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0), [языках программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) и т.п.

**Цель работы:** изучение алгоритмов поиска подстроки в строке.

**Задачи работы:**

1) изучение алгоритмов Бойера-Мура и Кнута-Морриса-Пратти

2) реализация этих алгоритмов

3) провести анализ и тестирование

# Аналитическая часть

В данной части дано теоретическое описание алгоритмов и указание области их применения.

# 1.1. Описание алгоритмов

Поиск подстроки в строке — одна из простейших задач поиска информации. Применяется в виде встроенной функции в текстовых редакторах, СУБД, поисковых машинах, языках программирования, программы определения плагиата осуществляют онлайн-проверку, используя алгоритмы поиска подстроки среди большого количества документов, хранящихся в собственной базе.

На сегодняшний день существует огромное разнообразие алгоритмов поиска подстроки. Программисту приходится выбирать подходящий в зависимости от таких факторов: длина строки, в которой происходит поиск, необходимость оптимизации, размер алфавита, возможность проиндексировать текст, требуется ли одновременный поиск нескольких строк. В данной лабораторной работе будут рассмотремы два алгоритма сравнения с образцом, алгоритм Кнута-Морриса-Пратта и алгоритм Бойера-Мура.

* + 1. **Стандартный алгоритм**

Стандартный алгоритм начинает со сравнения первого символа текста с первым символом подстроки. Если они совпадают, то происходит переход ко второму символу текста и подстроки. При совпадении сравниваются следующие символы. Так продолжается до тех пор, пока не окажется, что подстрока целиком совпала с отрезком текста, или пока не встретятся несовпадающие символы. В первом случае задача решена, во втором мы сдвигаем указатель текущего положения в тексте на один символ и заново начинаем сравнение с подстрокой

# 1.1.2 Алгоритм Бойера-Мура

Алгоритм Бойера-Мура осуществляет сравнение с образцом справа налево, а не слева направо. Исследуя искомый образец, можно осуществлять более эффективные прыжки в тексте при обнаружении несовпадения. В этом алгоритме кроме таблицы суффиксов применяется таблица стоп-символов. Она заполняется для каждого символа в алфавите. Для каждого встречающегося в подстроке символа таблица заполняется по принципу максимальной позиции символа в строке, за исключением последнего символа. При определении сдвига при очередном несовпадении строк, выбирается максимальное значение из таблицы суффиксов и стоп-символов

# Алгоритм Кнута-Морриса-Пратта

Алгоритм Кнута-Морриса-Пратта основан на принципе конечного автомата, однако он использует более простой метод обработки неподходящих символов. В этом алгоритме состояния помечаются символами, совпадение с которыми должно в данный момент произойти. Из каждого состояния имеется два перехода: один соответствует успешному сравнению, другой - несовпадению. Успешное сравнение переводит нас в следующий узел автомата, а в случае несовпадения мы попадаем в предыдущий узел, отвечающий образцу. В программной реализации этого алгоритма применяется массив сдвигов, который создается для каждой подстроки, которая ищется в тексте. Для каждого символа из подстроки рассчитывается значение, равное максимальной длине совпадающего префикса и суффикса относительно конкретного элемента подстроки. Создание этого массива позволяет при несовпадении строки сдвигать ее на расстояние, большее, чем 1 (в отличие от стандартного алгоритма).

# 1.2. Применение алгоритмов

Алгоритмы применяются в:

1. Системах проверки плагиата;
2. СУБД для поиска данных;
3. Текстовых редакторах

# Конструкторская часть

В данной части приведены схемы алгоритмов, а также их сравнительный анализ.

# 2.1 Требования к программе

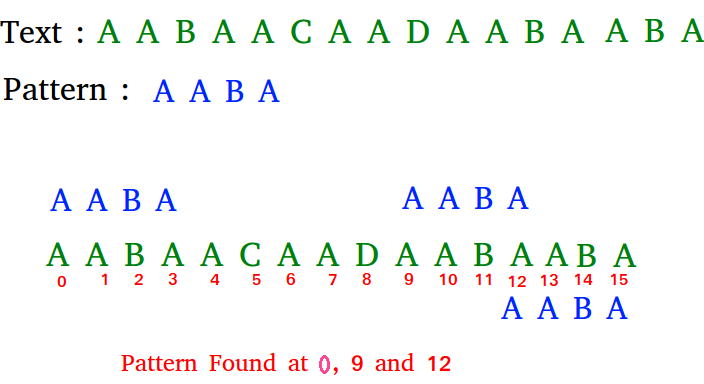
**Требования к вводу:** Длина подстроки должна быть меньше чем длина строки.

# 2.2 Пример работы алгоритмов

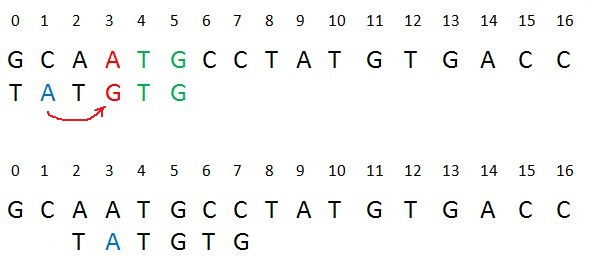
В рис. 1 и 2 будет рассмотрена работа алгоритмов КМП и Боейра-Мура на значениях строки text и подстроки pattern

text[]=”AABAACAADAABAABA” pattern[]=”AABA”

Работа алгоритма при КМП:



*Рисунок 1 – Работа алгоритма КМП*

Работа алгоритма БМ при плохом суффиксе:

*Рисунок 2 - в примере мы получили несоответствие в позиции 3. Здесь наш несовпадающий символ - «A». Теперь мы будем искать последнее вхождение «A» в паттерне. Мы получили «А» в позиции 1 в шаблоне (отображается синим цветом), и это последний случай его появления. Теперь мы сместим шаблон 2 раза, чтобы «А» в шаблоне выровнялось с «А» в тексте.*

# 2.3 Вывод

В данном разделе были рассмотрены основные требования к программе, показана работа алгоритмов на конкретной строке и подстроке.

# Технологическая часть

В данной части приведены используемые технические средства, а также примеры тестирования и листинг программы.

# 3.1 Требования к программному обеспечению

Программа должна корректно находить подстроку, верно показывать на её расположение. Требуется также обеспечить возможность замера времени работы каждого алгоритма на различных размерах строки.

# 3.2 Средства реализации

Выбран язык программирования Python3 за кроссплатформенность, автоматическое освобождение памяти, высокую скорость разработки. Замер процессорного времени проводился функцией perf\_counter() модуля time. Функция представлена на *листинге 1.*

***Листинг 1 – Текст на котором проводилось тестирование.***

txt **=** '''

from math import sin

sin(m)+

sin

asin(db)+

acos(rnrf)+

tan()+

prosto stroka tang()

just string but in english sqrt(1)+

sadece bir xett cos()+

cos

sqrt()+

sqrt

abs!

abs()+

abs)

abs[()] '''

# 3.3 Листинг кода

Исходный код программы приведен на *листингах 2,3,4.*

***Листинг 2* - Стандартный алгоритм поиска**

#Naive Pattern Searching algorithm

**def** search**(**pat**,** txt**):**

M **=** len**(**pat**)**

N **=** len**(**txt**)**

**for** i **in** range **(**N**-**M**+**1 **):**

j **=** 0

**while** **(**j **<** M**):**

**if** **(**txt**[**i**+**j**]** **!=** pat**[**j**]):**

**break**

j**+=**1

**if** **(**j **==** M**):**

**print** **(** "Pattern found at index " **,** i**)**

**print(**"End of search"**)**

***Листинг 3 – Алгоритм Боера-Мура Методом плохого суффикса***

NO\_OF\_CHARS **=** 256

**def** badCharHeuristic**(**string**,** size**):**

badChar **=** **[-**1**]\***NO\_OF\_CHARS

**for** i **in** range**(**size**):**

badChar**[**ord**(**string**[**i**])]** **=** i**;**

**return** badChar

**def** search**(**txt**,** pat**):**

'''

A pattern searching function that uses Bad Character

Heuristic of Boyer Moore Algorithm

'''

m **=** len**(**pat**)**

n **=** len**(**txt**)**

badChar **=** badCharHeuristic**(**pat**,** m**)**

s **=** 0

**while(**s **<=** n**-**m**):**

j **=** m**-**1

**while** j**>=**0 **and** pat**[**j**]** **==** txt**[**s**+**j**]:**

j **-=** 1

**if** j**<**0**:**

**print(**"Pattern occur at shift = {}"**.**format**(**s**))**

s **+=** **(**m**-**badChar**[**ord**(**txt**[**s**+**m**])]** **if** s**+**m**<**n **else** 1**)**

**else:**

s **+=** max**(**1**,** j**-**badChar**[**ord**(**txt**[**s**+**j**])])**

**print(**"End of search"**)**

***Листинг 4* - Алгоритм КМП**

**def** KMPSearch**(**pat**,** txt**):**

M **=** len**(**pat**)**

N **=** len**(**txt**)**

lps **=** **[**0**]\***M

j **=** 0 # index for pat[]

computeLPSArray**(**pat**,** M**,** lps**)**

i **=** 0 # index for txt[]

**while** i **<** N**:**

**if** pat**[**j**]** **==** txt**[**i**]:**

i **+=** 1

j **+=** 1

**if** j **==** M**:**

**print(**"Found pattern at index " **+** str**(**i**-**j**)** **)**

j **=** lps**[**j**-**1**]**

**elif** i **<** N **and** pat**[**j**]** **!=** txt**[**i**]:**

**if** j **!=** 0**:**

j **=** lps**[**j**-**1**]**

**else:**

i **+=** 1

**print(**"End of search"**)**

**def** computeLPSArray**(**pat**,** M**,** lps**):**

len **=** 0

lps**[**0**]**

i **=** 1

**while** i **<** M**:**

**if** pat**[**i**]==** pat**[**len**]:**

len **+=** 1

lps**[**i**]** **=** len

i **+=** 1

**else:**

**if** len **!=** 0**:**

len **=** lps**[**len**-**1**]**

**else:**

lps**[**i**]** **=** 0

i **+=** 1

# 3.4 Описание тестирования

Тестирование проводится по методу чёрного ящика [2]. Требуется проверить корректность работы.

# 3.5 Вывод

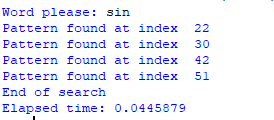
Текущая реализация на языке Python позволяет корректно находить заданную подстроку, а также производить замеры времени.

# Экспериментальная часть

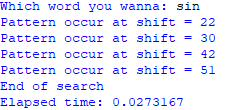
В этой части приведены пример интерфейса, входные данные тестирования, результаты замера времени и их анализ.

# 4.1. Примеры работы

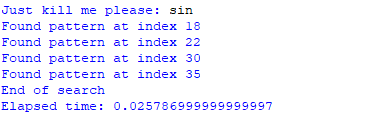
На *рис.* 3, 4, 5 приведены изображения внешнего вида интерфейса программы во время его работы.



*Рисунок 3 – Пример стандартной реализации*



*Рисунок 4 – Пример работы реализации Байера-Мура*

**

*Рисунок 5 – Пример работы реализации КПМ*

# 4.2. Результаты тестирования

В *таблице 1* представлены результаты тестирования по методу чёрного ящика [2] в следующем порядке: стандартный алгоритм, алгоритм Винограда, алгоритм Винограда с набором оптимизаций.

***Таблица 3***

**Результаты тестирования по методу черного ящика**

|  |  |  |  |
| --- | --- | --- | --- |
| Стандартная  реализация | Боейра-Мура | КПМ | Результат |
| sin | sin | sin | Ответ верный |
| cos | cos | cos | Ответ верный |
| sin( | sin( | sin( | Ответ верный |
| tan( | tan( | tan( | Ответ верный |
| sqrt( | sqrt( | sqrt( | Ответ верный |

Все тесты пройдены успешно.

# 4.3. Постановка эксперимента по замеру времени

Замер времени проводился для заданного текста. Один эксперимент повторялся не менее 5 раз, результат одного эксперимента рассчитывался как среднее значение результатов проведенных испытаний с одинаковыми входными данными.

4.4. Сравнительный анализ на материале экспериментальных данныхЗависимость времени нахождения подстроки в строке в зависимости от расположения представлен на *рисунках 6,7.\*

*Рисунок 6 – Время работы при ложных значениях.*

*Рисунок 7 – Время работы при поиске слова длиной 4 находящимся в тексте*

# 4.5 Вывод

Сравнительный анализ по времени показал, что стандартная реализация алгоритма очень нестабильна, а Бойера-Мура и КПМ показывают одинаковый результат.

# Заключение

В ходе лабораторной работы мы изучили возможности применения и реализовали алгоритмы поиска подстроки в строке.

Было проведено тестирование, показавшее, что алгоритмы реализованы правильно.

Временной анализ показал, что-то стандартный метод очень нестабилен и плохо работает при больших запросах.

# Список литературы

1. time — Time access and conversions // Python URL: https://docs.python.org/3/library/time.html (дата обращения: 1.11.2019).
2. Kara kutu test technologisi URL:<https://www.mobilhanem.com/kara-kutu-test-teknigi-ve-uygulanmasi/> (дата обращения 1.11.2019)
3. Bi̇lgi̇sayar tabanli si̇stemlerde test otomatizasyonunun tasarlanmasi ve gerçeklenmesi̇. Hacettepe Üniversitesi: 2015. 70 с.  
   URL:[shorturl.wizarmh/2lab](http://www.openaccess.hacettepe.edu.tr:8080/xmlui/bitstream/handle/11655/2713/63f38b48-243d-4811-8be9-bf09ae95593d.pdf?sequence=1)
4. Дж. Макконелл. Анализ алгоритмов. Активный обучающий подход.