*Федеральное государственное бюджетное образовательное учреждение* *высшего профессионального образования*

|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«Московский государственный технический университет  имени Н.Э. Баумана (национальный исследовательский институт)»***  ***(МГТУ им. Н.Э. Баумана)*** |

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА ИУ7

**Отчёт**

**по лабораторной работе №7**

**Дисциплина: Анализ алгоритмов**

**Тема лабораторной работы: Поиск образца в тексте**

Студент гр. ИУ7-51Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Громова В.П.** (Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Волкова Л.Л.**

(Подпись, дата) (И.О. Фамилия)

Москва, 2019г.

[**Введение**](#_l7rlvvzhhg0g) **3**

[**Аналитическая часть**](#_ymlsvkp7s19) **4**

[Стандартный алгоритм](#_iukg8u77v0qi) 4

[Алгоритм Кнута-Морриса-Пратта](#_wgkxsvz6vqwd) 6

[Алгоритм Бойера-Мура-Хорспула](#_m0f4cv40awtu) 8

[Вывод](#_l6miqyvs0wy8) 9

[**Конструкторская часть**](#_dwncbwgw3q7u) **11**

[Схемы алгоритмов](#_yenu5secabgz) 11

[**Технологическая часть**](#_i6e47vlkf5wy) **13**

[Требования к ПО](#_12dyot39kyjw) 13

[Средства реализации](#_3ht1resb1z6g) 13

[Листинги кода](#_1ivlund442c8) 13

[**Экспериментальная часть**](#_h9yzisnx544) **15**

[Разбор примеры для алгоритма КМП](#_c91cg94tkp99) 15

[Разбор примеров для алгоритма БМХ](#_489d0hmidqw7) 19

[Вывод](#_1522oeg4nt5f) 23

[**Заключение**](#_tifi50qojmn) **24**

## Введение

Поиск подстроки в строке (англ. String searching algorithm) — класс алгоритмов над строками, которые позволяют найти паттерн в тексте. Пусть есть некоторый текст Т и слово W. Необходимо найти первое (или все) вхождение(ия) этого слова в указанном тексте. Это действие типично для любых систем обработки текстов. Элементы массивов Т и W – символы некоторого конечного алфавита – например, 0, 1, или a, . . . , z, или а, . . . , я. Так как поиск подстроки в строке является одной из задач поиска информации, он применяется в виде встроенной функции в текстовых редакторах, СУБД, поисковых машинах, языках программирования и т. п.

На сегодняшний день существует разнообразие алгоритмов поиска подстроки. Программисту приходится выбирать подходящий в зависимости от таких факторов: длина строки, в которой происходит поиск, размер алфавита, возможность проиндексировать текст, частота повторяющихся символов. В данной лабораторной

работе рассмотрены основные алгоритмы сравнения с образцом: стандартный (примитивный) алгоритм, алгоритм Кнута-Морриса-Пратта, алгоритм Бойера-Мура-Хорспула.

Цель данной лабораторной работы: получить навык реализации алгоритмов сравнения с образцом.

Задачи лабораторной работы:

1. Описать алгоритмы
2. Реализовать алгоритмы
3. Разобрать пошагово показательные примеры работы алгоритмов
4. Сравнить эффективность

## Аналитическая часть

В данном разделе будут описаны такие алгоритмы поиска образца в строке, как стандартный алгоритм, алгоритм Кнута-Морриса-Пратта и алгоритм Бойера-Мура-Хорспула.

### Стандартный алгоритм

Стандартный алгоритм начинается со сравнения первого символа текста text с первым символом подстроки substr. 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 указатель текущего положения сдвигается в тексте на один символ и заново начинается сравнение с подстрокой. Далее в примере 1 будет показана работа алгоритма.

*Пример 1 - демонстрация работы стандартного алгоритма.*

Работа алгоритма для входной строки “abababcd” и шаблона “ababcd” продемонстрирована в таблице 1.1.1.

*Таблица 1.1.1 - Пример работы стандартного алгоритма.*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| text: | a | b | a | b | a | b | c | d |
| substr: | a | b | a | b | c | d |  |  |
| text: | a | b | a | b | a | b | c | d |
| substr: | a | b | a | b | c | d |  |  |
| text: | a | b | a | b | a | b | c | d |
| substr: | a | b | a | b | c | d |  |  |
| text: | a | b | a | b | a | b | c | d |
| substr: | a | b | a | b | c | d |  |  |
| text: | a | b | a | b | a | b | c | d |
| substr: | a | b | a | b | c | d |  |  |
| text: | a | b | a | b | a | b | c | d |
| substr: |  | a | b | a | b | c | d |  |
| text: | a | b | a | b | a | b | c | d |
| substr: |  |  | a | b | a | b | c | d |
| text: | a | b | a | b | a | b | c | d |
| substr: |  |  | a | b | a | b | c | d |
| text: | a | b | a | b | a | b | c | d |
| substr: |  |  | a | b | a | b | c | d |
| text: | a | b | a | b | a | b | c | d |
| substr: |  |  | a | b | a | b | c | d |
| text: | a | b | a | b | a | b | c | d |
| substr: |  |  | a | b | a | b | c | d |
| text: | a | b | a | b | a | b | c | d |
| substr: |  |  | a | b | a | b | c | d |

Худший случай работы алгоритма - когда при каждом проходе совпадают все символы кроме последнего. Пример такого случая - поиск вхождения подстроки “aaab” в текст “aaaaaaaaaaab”. Тогда сложность алгоритма равна O = S(T - S+1), где S - длина подстроки, T - длина текста. Таким образом, сложность пропорциональна O(length(text) \* length(str)).

Существуют несколько оптимизаций примитивного алгоритма, некоторые из которых будут рассмотрены далее.

### 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 означает переход в следующий узел автомата, а в несовпадения - попадание в предыдущий узел, отвечающий образцу. В программной реализации этого алгоритма применяется таблица сдвигов, создающаяся для каждой подстроки, которая ищется в тексте. Для каждого символа из подстроки рассчитывается значение, равное максимальной длине совпадающего префикса (начальные символы подстроки) и суффикса (конечные символы подстроки) относительно конкретного элемента подстроки. Для нулевого элемента подстроки смещение всегда равно нулю. Весь шаблон не рассматривается ни как префикс, ни как суффикс. Создание этого массива позволяет при несовпадении строки сдвигать ее на расстояние, большее, чем 1 (в отличие от стандартного алгоритма). В примере 2 будет показано формирование таблицы сдвигов.

*Пример 2 - формирование таблицы сдвигов.*

Для подстроки (шаблона) “abcabca” для каждого индекса вычисляется сдвиг. Для первой буквы “а” это ноль, далее так как символ “а” не равен символу “b”, то для первого символа “b” (с индексом 1) значение сдвига также равно нулю. Аналогично и для символа “с” (индекс 2). Для символа “а” с индексом 3 значение сдвига будет равно 1, так как “а” (индекс 1) равно “а” (индекс 3), однако далее “b” и “с” не равны. Таким образом продолжается вычисление для остальных индексов таблицы сдвигов (см. таблицу 1.2.1).

*Таблица 1.2.1 - Пример таблицы сдвигов.*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| индекс | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| шаблон | a | b | c | a | b | c | a |
| сдвиги | 0 | 0 |  |  |  |  |  |
| индекс | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| шаблон | a | b | c | a | b | c | a |
| сдвиги | 0 | 0 | 0 |  |  |  |  |
| индекс | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| шаблон | a | b | c | a | b | c | a |
| сдвиги | 0 | 0 | 0 | 1 |  |  |  |
| индекс | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| шаблон | a | b | c | a | b | c | a |
| сдвиги | 0 | 0 | 0 | 1 | 2 |  |  |
| индекс | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| шаблон | a | b | c | a | b | c | a |
| сдвиги | 0 | 0 | 0 | 1 | 2 | 3 |  |
| индекс | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| шаблон | a | b | c | a | b | c | a |
| сдвиги | 0 | 0 | 0 | 1 | 2 | 3 | 4 |

В последнем столбце значение сдвига для индекса 6 равно 4, так как префикс “abca” пересекается с суффиксом “abca” (символ “а”, выделенный голубым цветом, является одновременно концом префикса и началом суффикса). Таким образом, значение i-го сдвига соответствует индексу того символа в шаблоне, на который нужно переместиться в случае несовпадения на i-ом символе шаблона.

Алгоритм работает быстрее стандартного для строк с повторяющимися подстроками (за счет учета повторяющихся префиксов).

### Алгоритм Бойера-Мура-Хорспула

Алгоритм работает лучше предыдущего на случайных текстах. Отличия данного алгоритма от предыдущего состоят в том, что сканирование строк ведется справа налево и таблица сдвигов (смещений) вычисляется другим способом. Основная предварительная подготовка состоит в том, что нужно построить таблицу смещений для каждого символа. Таблица строится по следующим правилам: каждому символу ставится в соответствие величина, равная разности длины шаблона и порядкового номера символа (если символ повторяется, то берется самое правое вхождение). При этом если на каком-то шаге алгоритма последние символы не совпали, и символ, находящийся в исходной строке, не присутствует в шаблоне вообще, то нужно сдвинуть вправо на полную длину шаблона. Кроме того, последнему символу шаблона в таблице смещений присваивается значение длины шаблона, если больше этот символ в шаблоне не встречается, иначе присваивается значение этого же символа, следующего от конца. В пример 3 будет показано заполнение таблицы смещений.

*Пример 3 - заполнение таблицы смещений.*

Рассмотрим шаблон “dabcabca”. Таблица смещения (см. таблицу 1.3.1) заполняется как порядковый номер символа, если проходить шаблон справа налево. Однако первый в таком проходе символ имеет смещение не ноль, а либо длину всего шаблона, либо значение смещения, которое было присвоено ближайшему к нему такому же символу. В примере ниже символу “а” с индексом 7 было присвоено значение смещения равное 3, что соответствует смещению символа “а” с индексом 4. Так как символ “d” не встречался в строке кроме одного раза в самом начале шаблона, то его смещение соответствует длине шаблона - 1. Все остальные символы, не встретившиеся в шаблоне имеют смещение, равное длине шаблона, т.е. 8.

*Таблица 1.3.1 - Пример таблицы смещений.*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| индекс | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| шаблон | d | a | b | c | a | b | c | a |
| смещения | 7 | 3 | 2 | 1 | 3 | 2 | 1 | 3 |

После того, как таблица была составлена, исходная строка и шаблон совмещаются по началу, сравнение ведется по последнему символу. Если последние символы совпадают, то сравнение идет по предпоследнему символу и так далее. Если же символы не совпали, то шаблон смещается вправо, на число позиций взятое из таблицы смещений для символа из исходной строки, и тогда снова сравниваются последние символы исходной строки и шаблона. И так далее, пока или шаблон полностью не совпадает с подстрокой исходной строки, или не будет достигнут конец строки.

### Вывод

В данном разделе была рассмотрена задача поиска в подстроке, а также выделены основные алгоритмы ее решения - стандартный алгоритм, алгоритм Кнута-Морриса-Пратта (КМП) и алгоритм Бойера-Мура-Хорспула (БМХ). Стандартный алгоритм является универсальным вне зависимости от структуры текста, в то же время алгоритм совершает много сравнений, вследствие чего эффективность его падает. Алгоритм КМП предназначен для текстов с повторяющимися префиксами, в случае, если таковых нет, то этот алгоритм работает как и стандартный. Алгоритм БМХ также является универсальным, однако производит меньше сравнений, чем примитивный алгоритм, поэтому работает быстрее.

## Конструкторская часть

В данном разделе будут представлены схемы алгоритмов.

### Схемы алгоритмов

На рисунке 2.1.1 представлена схема работы алгоритма Кнута-Морриса-Пратта.

|  |
| --- |
|  |
| *Рис. 2.1.1 - схема алгоритма КМП* |

На рисунке 2.1.2 представлена схема работы алгоритма Бойера-Мура-Хорспула.

|  |
| --- |
|  |
| *Рис. 2.1.2 - схема алгоритма БМХ* |

## Технологическая часть

В данном разделе будут описаны требования к программе, средства реализации и представлены листинги кода. В данной лабораторной работе были реализованы алгоритмы КМП и БМХ.

### Требования к ПО

Программа на вход получает строку (search\_string) и искомую подстроку (pattern). Выход программы: индекс (индексы) вхождения подстроки либо сообщение о том, что подстрока не найдена.

### Средства реализации

Для реализации представленных алгоритмов был выбран язык Python.

### Листинги кода

В листинге 3.3.1 будет представлена реализация алгоритма КМП. Основная функция KMPsearch вызывает функцию compute\_array для вычисления массива сдвигов.

Листинг 3.3.1 - реализация алгоритма КМП.

1. **def KMPsearch(search\_string, pattern):**
2. search\_string.lower()
3. pattern.lower()
4. n = len(search\_string)
5. m = len(pattern)
6. shift\_array = compute\_array(pattern, m)
7. found = False
8. k = 0
9. p = 0
10. while k < n:
11. if pattern[p] == search\_string[k]:
12. k += 1
13. p += 1
14. if p == m:
15. found = True
16. print("KMP: match was found on index " + str(k - p) + "\n")
17. p = shift\_array[p - 1]
18. else:
19. if p == 0:
20. k += 1
21. if k > (n - m):
22. break
23. else:
24. p = shift\_array[p - 1]
25. if not found:
26. print("KMP: pattern not found")
27. **def compute\_array(pattern, m):**
28. shift\_array = [0] \* m
29. j = 0
30. i = 1
31. while i < m:
32. if pattern[i] == pattern[j]:
33. shift\_array[i] = j + 1
34. i += 1
35. j += 1
36. else:
37. if j == 0:
38. shift\_array[i] = 0
39. i += 1
40. else:
41. j = shift\_array[j - 1]
42. return shift\_array

В листинге 3.3.2 будет представлена реализация алгоритма БМХ. Основная функция BMHsearch вызывает функцию compute\_shift для вычисления таблицы смещений.

Листинг 3.3.2 - реализация алгоритма БМХ.

1. **def BMHsearch(search\_string, pattern):**
2. search\_string.lower()
3. pattern.lower()
4. n = len(search\_string)
5. m = len(pattern)
6. shift\_array = compute\_shift(pattern, m)
7. found = False
8. i = 0
9. while i <= (n - m):
10. j = m - 1
11. while (j >= 0) and pattern[j] == search\_string[i + j]:
12. j -= 1
13. if j < 0:
14. found = True
15. print("BMH: match was found on index " + str(i) + "\n")
16. i += (shift\_array[search\_string[i + m]] if i + m < n else 1)
17. else:
18. if shift\_array.get(search\_string[i + j]) is None:
19. i += m
20. else:
21. i += shift\_array[search\_string[i + j]]
22. if not found:
23. print("BMH: pattern not found.\n")
24. **def compute\_shift(pattern, m):**
25. shift\_array = {}
26. value = m - 1
27. for i in range(m - 1):
28. shift\_array[(pattern[i])] = value
29. value -= 1
30. return shift\_array

## Экспериментальная часть

В данном разделе рассмотрены пошаговые примеры выполнения алгоритмов поиска подстроки в строке (а именно алгоритмы КМП и БМХ).

### Разбор примеры для алгоритма КМП

В примере 4.1 будет показано действие алгоритма КМП, когда подстрока встречается в исходной строке поиска.

*Пример 4.1.1 - алгоритм КМП (1).*

Дана строка s = “abababcdabababa” и шаблон p = “ababa”. Сначала необходимо вычислить таблицу сдвигов t для шаблона (см. таблицу 4.1.1).

*Таблица 4.1.1 - пример таблицы сдвигов (1).*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 |
| p | a | b | a | b | a |
| t | 0 | 0 | 1 | 2 | 3 |

Теперь нужно начать сравнивать исходную строку с шаблоном слева направо. В первом столбце таблицы 4.1.2 указан номер прохода сравнений шаблона со строкой. Далее в первой строке указаны индексы исходной строки, во второй строке - сама строка, в которой осуществляется поиск, и далее указан шаблон.

*Таблица 4.1.2 - демонстрация работы алгоритма КМП (1).*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|  | s | a | b | a | b | a | b | c | d | a | b | a | b | a | b | a |
| 1 | p | a | b | a | b | a |  |  |  |  |  |  |  |  |  |  |
| 2 | p |  |  | a | b | a | b | a |  |  |  |  |  |  |  |  |
| 3 | p |  |  |  |  | a | b | a | b | a |  |  |  |  |  |  |
| 4 | p |  |  |  |  |  |  | a | b | a | b | a |  |  |  |  |
| 5 | p |  |  |  |  |  |  |  | a | b | a | b | a |  |  |  |
| 6 | p |  |  |  |  |  |  |  |  | a | b | a | b | a |  |  |
| 7 | p |  |  |  |  |  |  |  |  |  |  | a | b | a | b | a |

Пусть i - текущий индекс строки, символ с этим индексом сравнивается; j - индекс символа шаблона, с которым ведется сравнение символа строки.

Проходы:

1. Все символы подстроки и части исходной строки совпали, это значит, что вхождение подстроки в строку найдено, индекс начала этого вхождения i = 0. Далее смещаем шаблон так, чтобы сравнение шаблона началось с того индекса, значение которого равно значению сдвига последнего символа в шаблоне (т.е. i = 5, j = 3), чтобы продолжить поиск.
2. Символы “c” и “а” не совпали, на следующем проходе будем сравнивать символы шаблона, начиная с индекса 2 (т.е. j = 2, i не изменилось и = 6).
3. Символы “c” и “а” опять не совпали, на следующем проходе будем сравнивать символы шаблона, начиная с индекса 0 (т.е. j = 0, i не изменилось и = 6).
4. Символы “c” и “а” опять не совпали, на следующем проходе будем сравнивать символы шаблона, начиная с индекса 0, но при этом будем значение индекса исходной строки увеличим на 1 (т.е. j = 0, i = 7).
5. Символы “d” и “а” не равны, сравниваем шаблон с самого начала а индекс строки увеличиваем на 1 (т.е. j = 0, i = 8).
6. Все символы шаблона совпали с соответствующими символами исходной строки, это значит, что еще одно вхождение найдено, теперь уже индекс этого вхождения i = 8. Далее индекс шаблона равен 3 (как и в случае после первого прохода): j = 3, i = 13.
7. Все символы снова совпали, мы нашли еще одно вхождение при значении индекса исходной строки i = 10.

Итого были найдены все 3 вхождения подстроки “ababa” в строку “abababcdabababa”.

В примере 4.1.2 будет показано действие алгоритма КМП, когда подстрока не встречается в исходной строке поиска.

*Пример 4.1.2 - алгоритм КМП (2).*

Дана строка s = “ababab” и шаблон p = “acd”. Сначала необходимо вычислить таблицу сдвигов t для шаблона (см. таблицу 4.1.3).

*Таблица 4.1.3- пример таблицы сдвигов (2).*

|  |  |  |  |
| --- | --- | --- | --- |
| i | 0 | 1 | 2 |
| p | a | c | d |
| t | 0 | 0 | 0 |

Теперь нужно начать сравнивать исходную строку с шаблоном слева направо. В первом столбце таблицы 4.1.4 указан номер прохода сравнений шаблона со строкой. Далее в первой строке указаны индексы исходной строки, во второй строке - сама строка, в которой осуществляется поиск, и далее указан шаблон.

*Таблица 4.1.4 - демонстрация работы алгоритма КМП (2).*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | i | 0 | 1 | 2 | 3 | 4 | 5 |
|  | s | a | b | a | b | a | b |
| 1 | p | a | c | d |  |  |  |
| 2 | p |  | a | c | d |  |  |
| 3 | p |  |  | a | c | d |  |
| 4 | p |  |  |  | a | c | d |

Пусть i - текущий индекс строки, символ с этим индексом сравнивается; j - индекс символа шаблона, с которым ведется сравнение символа строки.

Проходы:

1. Символы “а” совпали, но символы “b” и “с” не совпали, поэтому смещаем шаблон так, чтобы сравнение шаблона началось с того индекса, значение которого равно значению сдвига символа “а” в шаблоне (т.е. i = 1, j = 0).
2. Символы “b” и “а” не совпали, на следующем проходе будем сравнивать символы шаблона, начиная с индекса 0, а индекс исходной строки увеличиваем на 1 (т.е. j = 0, i = 2).
3. Символы “а” совпали, но символы “b” и “с” не совпали, на следующем проходе будем сравнивать символы шаблона, начиная с индекса 0, а индекс исходной строки увеличиваем на 1 (т.е. j = 0, i = 3).
4. Символы “b” и “c” не совпали, на следующем проходе будем сравнивать символы шаблона, начиная с индекса 0, значение индекса исходной строки увеличим на 1 (т.е. j = 0, i = 4).

Далее сравнения производится не будут, потому что количество символов в исходной строке, которые еще не были обработаны, меньше, чем длина шаблона.

### Разбор примеров для алгоритма БМХ

В примере 4.2.1 будет показано действие алгоритма БМХ, когда подстрока встречается в исходной строке поиска.

*Пример 4.2.1 - алгоритм БМХ (1).*

Дана строка s = “abababcdabababa” и шаблон p = “ababa”. Сначала необходимо вычислить таблицу смещения t для символов шаблона (см. таблицу 4.2.1).

*Таблица 4.2.1 - пример таблицы смещений (1).*

|  |  |  |  |
| --- | --- | --- | --- |
| p | a | b | \* |
| t | 2 | 1 | 5 |

Так как символ “b” стоит сразу после последнего символа шаблона, он имеет смещение 1, далее идеи символ “а”, поэтому его смещение равно 2. Все остальные символы это или символы “а”, или символы “b”, минимальные смещения (относительно конца шаблона) для которых уже вычислены. Символом “\*” в таблице 4.2.1 отмечены все остальные символы, которые не встречаются в шаблоне, для них смещение равно длине шаблона, т.е. 5.

Далее в таблице 4.2.2 будет рассмотрен непосредственно поиск шаблона p в строке s.

*Таблица 4.2.2 - демонстрация работы алгоритма БМХ (1).*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|  | s | a | b | a | b | a | b | c | d | a | b | a | b | a | b | a |
| 1 | p | a | b | a | b | a |  |  |  |  |  |  |  |  |  |  |
| 2 | p |  | a | b | a | b | a |  |  |  |  |  |  |  |  |  |
| 3 | p |  |  | a | b | a | b | a |  |  |  |  |  |  |  |  |
| 4 | p |  |  |  |  |  |  |  | a | b | a | b | a |  |  |  |
| 5 | p |  |  |  |  |  |  |  |  | a | b | a | b | a |  |  |
| 6 | p |  |  |  |  |  |  |  |  |  | a | b | a | b | a |  |
| 7 | p |  |  |  |  |  |  |  |  |  |  | a | b | a | b | a |

Пусть i - текущий индекс строки, символ с этим индексом сравнивается; j - индекс символа шаблона, с которым ведется сравнение символа строки. В начале каждого прохода j = m - 1, где m - длина шаблона, т.к. сравнение производится справа налево. Сравниваются символы с индексами i + j для исходной строки и j для шаблона.

Проходы:

1. Все символы подстроки и части исходной строки совпали, это значит, что вхождение подстроки в строку найдено, индекс начала этого вхождения i = 0. Далее индекс исходной строки равен значению смещения для символа при i = 5, т.е. “b”, теперь i = 1.
2. Символы “b” и “а” не совпали, поэтому смещаем индекс исходной строки на значение смещения для символа “b”, т.е. i = 1 + 1 = 2.
3. Символы “c” и “а” не совпали, поэтому смещаем индекс исходной строки на значение смещения для символа “c”, т.е. i = 2 + 5 = 7.
4. Символы “b” и “а” не совпали, поэтому смещаем индекс исходной строки на значение смещения для символа “b”, т.е. i = 7 + 1 = 8.
5. Все символы подстроки и части исходной строки совпали, это значит, что вхождение подстроки в строку найдено, индекс начала этого вхождения i = 8. Далее индекс исходной строки равен значению смещения для символа при i = 13, т.е. “b”, теперь i = 8 + 1= 9.
6. Символы “b” и “а” не совпали, поэтому смещаем индекс исходной строки на значение смещения для символа “b”, т.е. i = 9 + 1 = 10.
7. Все символы подстроки и части исходной строки совпали, это значит, что вхождение подстроки в строку найдено, индекс начала этого вхождения i = 10. Сравнение дальше проводить нет смысла, поскольку количество символов в исходной строке, которые еще не были обработаны, меньше, чем длина шаблона.

Итого были найдены все 3 вхождения подстроки “ababa” в строку “abababcdabababa”.

В примере 4.2.2 будет показано действие алгоритма БМХ, когда подстрока не встречается в исходной строке поиска.

*Пример 4.2.2 - алгоритм БМХ (2).*

Дана строка s = “ababab” и шаблон p = “acd”. Сначала необходимо вычислить таблицу смещения t для символов шаблона (см. таблицу 4.2.3).

*Таблица 4.2.3 - пример таблицы смещений (2).*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| p | a | c | d | \* |
| t | 2 | 1 | 3 | 3 |

Так как символ “c” стоит сразу после последнего символа шаблона, он имеет смещение 1, далее идеи символ “а”, поэтому его смещение равно 2. Так как символ “d” встречает только один раз и в конце шаблона, то его смещение равно длине шаблона. Символом “\*” в таблице 4.2.3 отмечены все остальные символы, которые не встречаются в шаблоне, для них смещение равно длине шаблона, т.е. 3.

Далее в таблице 4.2.4 будет рассмотрен непосредственно поиск шаблона p в строке s.

*Таблица 4.2.4 - демонстрация работы алгоритма БМХ (2).*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | i | 0 | 1 | 2 | 3 | 4 | 5 |
|  | s | a | b | a | b | a | b |
| 1 | p | a | с | d |  |  |  |
| 2 | p |  |  | a | c | d |  |

Пусть i - текущий индекс строки, символ с этим индексом сравнивается; j - индекс символа шаблона, с которым ведется сравнение символа строки. В начале каждого прохода j = m - 1, где m - длина шаблона, т.к. сравнение производится справа налево. Сравниваются символы с индексами i + j для исходной строки и j для шаблона.

Проходы:

1. Символы “a” и “d” не совпали, поэтому смещаем индекс исходной строки на значение смещения для символа “a”, т.е. i = 0 + 2 = 2.
2. Символы “a” и “d” не совпали, поэтому смещаем индекс исходной строки на значение смещения для символа “a”, т.е. i = 2 + 2 = 4. Сравнение дальше проводить нет смысла, поскольку количество символов в исходной строке, которые еще не были обработаны, меньше, чем длина шаблона.

### Вывод

Как видно из приведенных примеров, алгоритм Кнута-Морриса-Пратта работает быстрее стандартного алгоритма в случае присутствия повторяющихся префиксов в шаблоне, однако если повторяющихся префиксов нет, то алгоритм начинает работать как примитивный, что хорошо видно из примера 4.1.2. Также из рассмотренных примеров видно, что алгоритм Бойера-Мура-Хорспула лучше справляется с задачей поиска подстроки в строке с произвольной структурой, то есть вне зависимости от того, есть ли там повторяющиеся префиксы или нет.

## Заключение

В ходе работы были изучены и реализованы основные алгоритмы для решения задачи поиска подстроки в строке: стандартный алгоритм, алгоритм Бойера-Мура-Хорспула и алгоритм Кнута-Морриса-Пратта. Был проведен их сравнительный анализ, в ходе которого были выявлены лучшие и худшие случаи использования каждого из них. В результате стандартный (примитивный) алгоритм является универсальным, не требует предварительной обработки, однако производит лишние сравнения. В то же время алгоритм Бойера-Мура-Хорспула работает одинаково на всех типах строк, то есть он также универсален, в среднем производит меньше сравнений, чем примитивный алгоритм, за счет нескольких модификаций. При этом для этого алгоритма необходима предварительная обработка (сложность O(len(pattern))). Однако в худшем случае алгоритм будет иметь ту же сложность, что и примитивный. Алгоритм Кнута-Морриса-Пратта имеет линейную сложность алгоритма (O(len(text) + len(pattern)) для строк, содержащих повторяющиеся префиксы. Однако для произвольных текстов его сложность может вырасти до сложности примитивного алгоритма. Таким образом, для случайных входных строк оптимальным алгоритмом будет алгоритм Бойера-Мура-Хорспула.