*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 подстроки в строке**

Студент гр. ИУ7-51Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Зейналов З. Г** (Подпись, дата) (И.О. Фамилия)

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

(Подпись, дата) (И.О. Фамилия)

Москва, 2019г.

[Введение 3](#_Toc27679142)

[1. Аналитическая часть 4](#_Toc27679143)

[1.1. Задача поиска подстроки в строке 4](#_Toc27679144)

[1.2. Стандартный алгоритм 4](#_Toc27679145)

[Вывод 6](#_Toc27679146)

[2. Конструкторская часть 7](#_Toc27679147)

[2.1. Пример работы алгоритмов 7](#_Toc27679148)

[2.2. Алгоритм Кнутта-Мориса-Пратта 7](#_Toc27679149)

[2.3. Алгоритм Бойера-Мура 7](#_Toc27679150)

[3. Технологическая часть 8](#_Toc27679151)

[3.1. Средства реализации 8](#_Toc27679152)

[3.2. Листинги кода 8](#_Toc27679153)

[3.3. Вывод 10](#_Toc27679154)

[4. Экспериментальная часть 11](#_Toc27679155)

[4.1. Примеры работы программы 11](#_Toc27679156)

[4.2. Постановка эксперимента 11](#_Toc27679157)

[4.3. Сравнительный анализ на материалах эксперимента 12](#_Toc27679158)

[4.4. Вывод 12](#_Toc27679159)

[Заключение 13](#_Toc27679160)

### Введение

Цель данной лабораторной работы: Получить навык реализации алгоритмов сравнения с образцом на примере алгоритмов на основании автоматов.

Задачи лабораторной работы следующие.

1. Изучение и описание алгоритмов.
2. Разработка и реализация алгоритмов.
3. Тестирование полученного программного обеспечения

### 

### Аналитическая часть

В данном разделе описаны алгоритмы поиска подстроки в строке.

#### Задача поиска подстроки в строке

Пусть дана некоторая строка (образец) S и некоторый текст T. Необходимо определить является ли образец подстрокой строки S. Длина образца меньше длины строки. Если алгоритм поиска подстроки в строке обнаруживает вхождение S в T, то он должен вернуть индекс, указывающий на первое вхождение.

#### Стандартный алгоритм

Простейшим алгоритмом является стандартный алгоритм, принцип которого включает в себя проход по всем символам исходной строки и посимвольного сравнения с символами образца, до тех пор, пока не образец не будет найден в исходной строке или исходная строка не будет исчерпана. Сложность данного алгоритма при длинах строк n – длина образца, m – длина исходной строки равна O(n\*m)

* 1. **Алгоритм Кнута-Морриса-Пратта**

Алгоритм Кнута-Морриса-Пратта позволяет улучшить показатель количества сравнений: данный алгоритм требует только N сравнений в худшем случае. Идея алгоритма в том, что при каждом несовпадении T[I] и W[J] мы сдвигаемся не на единицу, а на J, так как меньшие сдвиги не приведут к полному совпадению. К сожалению, этот алгоритм поиска дает выигрыш только тогда, когда несовпадению предшествовало некоторое число совпадений, иначе алгоритм работает как примитивный. Так как совпадения встречаются реже, чем несовпадения, выигрыш в большинстве случаев незначителен. Алгоритм Кнута-Морриса-Пратта основан на принципе конечного автомата. В этом алгоритме состояния помечаются символами, совпадение с которыми должно в данный момент произойти. Из каждого состояния имеется два перехода: один соответствует успешно-му сравнению, другой — несовпадению. Успешное сравнение переводит нас в следующий узел автомата, а в случае несовпадения мы попадаем в предыдущий узел, отвечающий образцу. При всяком переходе по успешному сравнению в конечном автомате Кнута-Морриса-Пратта происходит выборка нового символа из текста. Переходы, отвечающие неудачному сравнению, не приводят к выборке нового символа; вместо этого они повторно используют последний выбранный символ. Если мы перешли в конечное состояние, то это означает, что искомая подстрока найдена. Заметим, что при совпадении ничего особенного делать не надо: происходит переход к следующему узлу. Напротив, переходы по несовпадению определяются тем, как искомая подстрока соотносится сама с собой. Метод КМП использует предобработку искомой строки, а именно: на ее основе создается префикс-функция. Префикс-функция от строки S и позиции i в ней — длина k наибольшего собственного (не равного всей подстроке) префикса подстроки S[1..i], который одновременно является суффиксом этой подстроки. То есть, в начале подстроки S[1..i] длины i нужно найти такой префикс максимальной длины k < i, который был бы суффиксом данной подстроки S[1..k] = S[(i − k + 1)..i]. Например, для строки "abcdabscabcdabia" префикс-функция будет такой: [0 ,0 ,0 ,0 ,1 ,2 ,0 ,0 ,1 ,2 ,3 ,4 ,5 ,6 ,0 ,1]. Значения префикс-функции для каждого символа шаблона вычисляются перед началом поиска подстроки в строке и затем используются для сдвига. Особенностью данного алгоритма является то, что он работает на основе автоматов. Так, например, для нахождения в строке "abababcb" подстроки "ababcb" мы построим следующий автомат 1, где состояния маркируются ожидаемыми символами:

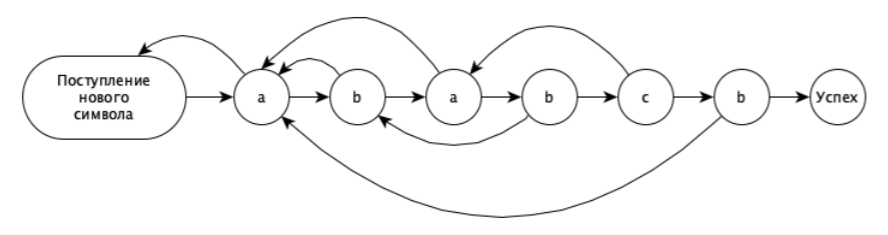


Рисунок 1 - Автомат

Также существует оптимизация алгоритма:

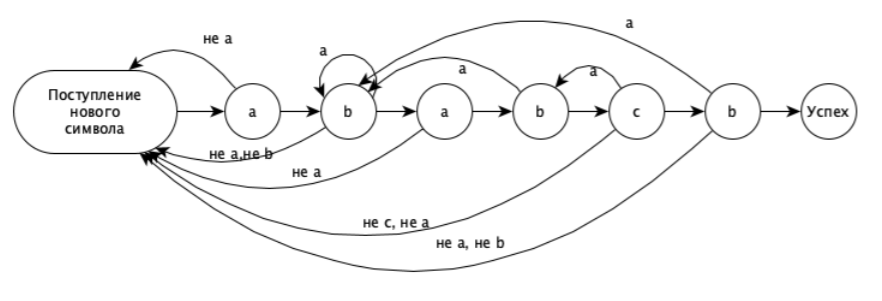


Рисунок 2 - Оптимизированный автомат

* 1. **Алгоритм Бойера-Мура**

Алгоритм поиска строки Бойера — Мура считается наиболее быстрым среди алгоритмов общего назначения, предназначенных для поиска подстроки в строке. Преимущество этого алгоритма в том, что ценой некоторого количества предварительных вычислений над шаблоном (но не над строкой, в которой ведётся поиск) шаблон сравнивается с исходным текстом не во всех позициях — часть проверок пропускаются как заведомо не дающие результата. Идея БМ-поиска – сравнение символов начинается с конца образца, а не с начала, то есть сравнение отдельных символов происходит справа налево. Затем с помощью некоторой эвристической процедуры вычисляется величина сдвига вправо s. И снова производится сравнение символов, начиная с конца образца. Простейший вариант алгоритма Бойера-Мура состоит из следующих шагов. На первом шаге мы строим таблицу смещений для искомого образца. Процесс построения таблицы будет описан ниже. Далее мы совмещаем начало строки и образца и начинаем проверку с последнего символа образца. Если последний символ образца и соответствующий ему при наложении символ строки не совпадают, образец сдвигается относительно строки на величину, полученную из таблицы смещений, и снова проводится сравнение, начиная с последнего символа образца. Если же символы совпадают, производится сравнение предпоследнего символа образца и т. д. Если все символы образца совпали с наложенными символами строки, значит, мы нашли подстроку и поиск окончен. Если же какой-то (не последний) символ образца не совпадает с соответствующим символом строки, мы сдвигаем образец на один символ вправо и снова начинаем проверку с последнего символа. Весь алгоритм выполняется до тех пор, пока либо не будет найдено вхождение искомого образца, либо не будет достигнут конец строки. Таблица смещений строится следующим образом. Каждому символу ставится в соответствие величина, равная разности длины шаблона и порядкового номера символа (если символ повторяется, то берется самое правое вхождение). Величина смещения для каждого символа образца зависит только от порядка символов в образце, поэтому смещения удобно вычислить заранее и хранить в виде одномерного массива, где каждому символу алфавита соответствует смещение относительно последнего символа образца.

#### Вывод

В данном разделе были рассмотрены основные алгоритмы поиска образца в строке.

### Конструкторская часть

В данном разделе будут представлены схемы алгоритмов.

#### Пример работы алгоритмов

В таблице 1 и таблице 2 будут рассмотрены примеры работы алгоритмов Кнута-Морриса-Пратта и Бойера-Мура на входных данных строки s и подстроки sub:

*String sub = “abaa”, string s = “ababacabaa”.*

#### Алгоритм Кнута-Морриса-Пратта

Для алгоритма Кнута-Морриса-Пратта вычисленный массив префиксов для заданной строки sub выглядит так: prefix = [0, 0, 1, 1]. Таблица 1 отображает пошаговую работу алгоритма при полученном массиве префиксов.

*Таблица 1 - Пошаговая работа алгоритма Кнута-Морриса-Пратта*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | a | b | a | c | a | b | a | a |
| a | b | a | a |  |  |  |  |  |  |
|  |  | a | b | a | a |  |  |  |  |
|  |  |  |  | a | b | a | a |  |  |
|  |  |  |  |  | a | b | a | a |  |
|  |  |  |  |  |  | a | b | a | a |

#### Алгоритм Бойера-Мура

Для алгоритма Бойера-Мура вычисленный массив суффиксов для заданной подстроки sub имеет значение: suffix = [2, 5, 5, 6]. Переходы алфавита для подстроки sub: letters = [‘a’ = 0, ‘b’ = 2] Если буквы нет в letters, будет считаться, что переход равен длине sub.

Таблица 2 - пошаговая работа алгоритма Бойера-Мура

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | a | b | a | c | a | b | a | a |
| a | b | a | a |  |  |  |  |  |  |
|  |  | a | b | a | a |  |  |  |  |
|  |  |  |  |  |  | a | b | a | a |

### Технологическая часть

В этом разделе будут описаны требования к программе, средства реализации и представлены листинги кода.

#### Средства реализации

Для реализации вышеописанных алгоритмов был выбран язык программирования C++. Были использованы класс vector из стандартной библиотеки. Среда разработки: QtCreator.

#### Листинги кода

Листинг 1: Простой алгоритм.

1. int **baseAlgo**(*const* string substr, *const* string str)
2. {
3. unsigned int strLen = str.size();
4. unsigned int substrLen = substr.size();
5. int index = -1;
6. *if* (strLen >= substrLen)
7. {
8. int count = 0;
9. *for* (int i = 0; i < strLen; i++)
10. {
11. count = 0;
12. *for* (int j = 0; j < substrLen; j++)
13. {
14. *if* (str[i + j] == substr[j])
15. {
16. count++;
17. }
18. }
19. *if* (count == substrLen)
20. {
21. index = i;
22. *break*;
23. }
24. }
25. }
26. *return* index;
27. }

В листинге 2 представлена реализация алгоритма Кнутта-Морисса-Пратта. Листинг 2: Реализация алгоритма Кнутта-Морриса-Пратта.

1. vector<size\_t> **prefix\_function** (string s)
2. {
3. size\_t n = s.length();
4. vector<size\_t> pi(n);
5. *for* (size\_t i = 1; i < n; ++i)
6. {
7. size\_t j = pi[i - 1];
8. *while* ((j > 0) && (s[i] != s[j]))
9. j = pi[j - 1];
10. *if* (s[i] == s[j])
11. ++j;
12. pi[i] = j;
13. }
14. *return* pi;
15. }
16. }
17. int **KMP**(*const* string substr, *const* string str)
18. {
19. vector<size\_t> v = prefix\_function(str);
20. int result = -1;
21. int i = 0;
22. int j = 0;
23. *while* (i < str.length())
24. {
25. *if* (substr[j] == str[i])
26. {
27. j++;
28. i++;
29. }
30. *if* (j == substr.length())
31. {
32. result = i - j;
33. j = v[j - 1];
34. }
35. *else* *if* (i < str.length() && substr[j] != str[i])
36. {
37. *if* (j != 0)
38. {
39. j = v[j - 1];
40. }
41. *else*
42. {
43. i++;
44. }
45. }
46. }
47. *return* result;
48. }

В листинге 3 представлена реализация алгоритма Бойера-Мура.

Листинг 3: Реализация алгоритма Бойера-Мура.

1. void **badCharHeuristic**(string str, int badchar[NO\_OF\_CHARS])
2. {
3. int i;
4. int size = str.length();
5. *for* (i = 0; i < NO\_OF\_CHARS; i++)
6. badchar[i] = -1;
7. *for* (i = 0; i < size; i++)
8. badchar[(int) str[i]] = i;
9. }
10. int **BM**(*const* string pat, *const* string txt)
11. {
12. int m = pat.length();
13. int n = txt.length();
14. int badchar[NO\_OF\_CHARS];
15. badCharHeuristic(pat, badchar);
16. int s = 0;
17. *while* (s <= (n - m))
18. {
19. int j = m - 1;
20. *while* (j >= 0 && pat[j] == txt[s + j])
21. j--;
22. *if* (j < 0)
23. {
24. *return* s;
25. s += (s + m < n) ? m - badchar[txt[s + m]] : 1;
26. }
27. *else*
28. s += max(1, j - badchar[txt[s + j]]);
29. }
30. }

#### Вывод

Были реализованы три алгоритма, решающие задачу нахождения образца в строке, для дальнейшего сравнительного анализа точности и скорости вычислений.

### Экспериментальная часть

В данном разделе будет проведен временной анализ работы алгоритмов..

#### Примеры работы программы

На рисунках 3 - 4 будут приведены примеры работы программы. На вход поступают 2 строки. На выходе – результаты работы 3 алгоритмов – значение индекса, начиная с которого строка sub входит в строку s;

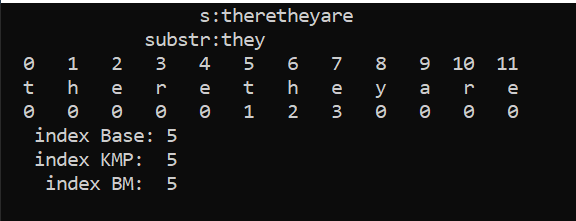


Рисунок 3 - Пример работы 1

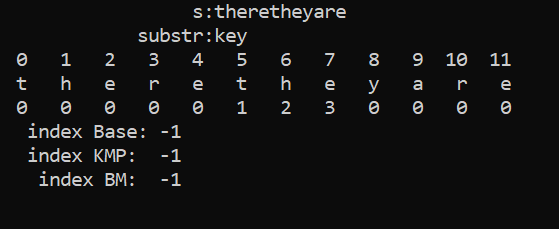


Рисунок 4 - Пример работы алгоритма Бойера-Мура

Как видно из приведенных примеров, Результаты программы совпадают с ожидаемыми, а именно при не вхождении подстроки в строку возвращается результат -1, иначе возвращается результат равный индексу. .При функциональном тестировании все тесты были пройдены и результаты совпали с ожидаемыми.

#### Постановка эксперимента

#### Сравнительный анализ на материалах эксперимента

Был проведен замер времени работы алгоритмов при разных размерах строки и фиксированном размере подстроки.

#### Вывод

### Заключение

В ходе лабораторной работы были изучены и реализованы алгоритмы поиска подстроки в строке, проанализированы возможности применения.

Было проведено тестирование, показавшее, что алгоритмы реализованы правильно.