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»

Отчет
по лабораторной работе № 8

Дисциплина: Анализ алгоритмов

Тема: Поиск подстроки в строке

Студент: Платонова Ольга

Группа: ИУ7-55Б

Преподаватели: Волкова Л. Л.
Строганов Ю. В.

Москва, 2020 г.

Оглавление

Введение

В данной лабораторной работе требуется изучить и реализовать алгоритмы поиска подстроки в строке. Так, требуется реализовать стандартный алгоритм поиска, алгоритм Кнута-Морриса-Пратта и алгоритм Бойера-Мура с не менее, чем 2 эвристиками. Также необходимо выполнить сравнительный анализ времени работы для каждой из реализаций.

1. Аналитическая часть

В данном разделе будут рассмотрены понятия, связанные с поиском подстроки и алгоритмами, его реализующими.

1.1. Задача поиска подстроки

Поиск подстроки в строке — одна из простейших задач поиска информации. Пусть есть некоторый текст T и слово (или образ) W. Необходимо найти первое вхождение этого слова в указанном тексте. Это действие типично для любых систем обработки текстов. Элементы массивов T и W – символы некоторого конечного алфавита – например, {0, 1}, или {a, …, z}, или {а, …, я}.

Наиболее типичным приложением такой задачи является документальный поиск: задан фонд документов, состоящих из последовательности библиографических ссылок, каждая ссылка сопровождается «дескриптором», указывающим тему соответствующей ссылки. Надо найти некоторые ключевые слова, встречающиеся среди дескрипторов. [1]

Поиск строки формально определяется следующим образом. Пусть задан массив T из N элементов и массив W из M элементов, причем $0<M\leq N$. Поиск строки обнаруживает первое вхождение W в T, результатом будем считать индекс i, указывающий на первое с начала строки (с начала массива T) совпадение с образом (словом).

На сегодняшний день существует огромное разнообразие алгоритмов поиска подстроки.

1.2. Стандартный алгоритм поиска

Стандартный алгоритм является простейшей реализацией решения задачи поиска подстроки. Алгоритм требует малые трудозатраты на программу и совсем не требует памяти. Недостатками алгоритма являются:

1. высокая сложность — O(N*M), в худшем случае – Θ((N-M+1)*M);
2. после несовпадения просмотр всегда начинается с первого символа образца и поэтому может включать символы T, которые ранее уже просматривались;
3. информация о тексте T, получаемая при проверке данного сдвига S, никак не используется при проверке последующих сдвигов.

Доказано, что стандартный алгоритм отрабатывает в среднем 2h сравнений. [2]

1.3. Алгоритм Кнута-Морриса-Пратта

Этот алгоритм появился в результате тщательного анализа стандартного алгоритма. Исследователи хотели найти способы более полно использовать информацию, полученную во время сканирования (стандартный алгоритм ее просто отбрасывает). Поэтому для работы алгоритма требуется предварительное вычисление префикс-функции.

Время работы алгоритма линейно зависит от объёма входных данных, то есть разработать асимптотически более эффективный алгоритм невозможно.

Особенности алгоритма КМП-поиска:

1. требуется порядка (N+M) сравнений символов для получения результата;
2. схема КМП-поиска дает подлинный выигрыш только тогда, когда неудаче предшествовало некоторое число совпадений. Лишь в этом случае образ сдвигается более чем на единицу. К несчастью, совпадения встречаются значительно реже чем несовпадения. Поэтому выигрыш от КМП-поиска в большинстве случаев текстов весьма незначителен. [3]

1.4. Алгоритм Бойера-Мура

Алгоритм поиска Бойера — Мура — алгоритм общего назначения, предназначенный 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.

Алгоритм основан на трёх идеях.

1. Сканирование слева направо, сравнение справа налево. Совмещается начало текста (строки) и шаблона, проверка начинается с последнего символа шаблона. Если символы совпадают, производится сравнение предпоследнего символа шаблона и т. д. Если все символы шаблона совпали с наложенными символами строки, значит, подстрока найдена, и выполняется поиск следующего вхождения подстроки.

2. Эвристика стоп-символа. Данная эвристика для своей работы требует дополнительную память и дополнительное время на этапе подготовки шаблона.

3. Эвристика совпавшего суффикса. Неформально, если при чтении шаблона справа налево совпал суффикс S, а символ b, стоящий перед S в шаблоне (то есть шаблон имеет вид

PbS), не совпал, то эвристика совпавшего суффикса сдвигает шаблон на наименьшее число позиций вправо так, чтобы строка S совпала с шаблоном, а символ, предшествующий в шаблоне данному совпадению S, отличался бы от b (если такой символ вообще есть). [4]

1.5. Вывод

В данном разделе были рассмотрены понятия, связанные с поиском подстроки и разобраны алгоритмы, реализующие поиск: стандартный, Кнута-Морриса-Пратта и Бойера-Мура.

2. Конструкторская часть

В данном разделе будут рассмотрены схемы для каждой из реализаций алгоритма поиска подстроки.

2.1. Разработка алгоритмов

На рисунке 1 изображена схема *стандартного алгоритма* поиска подстроки.

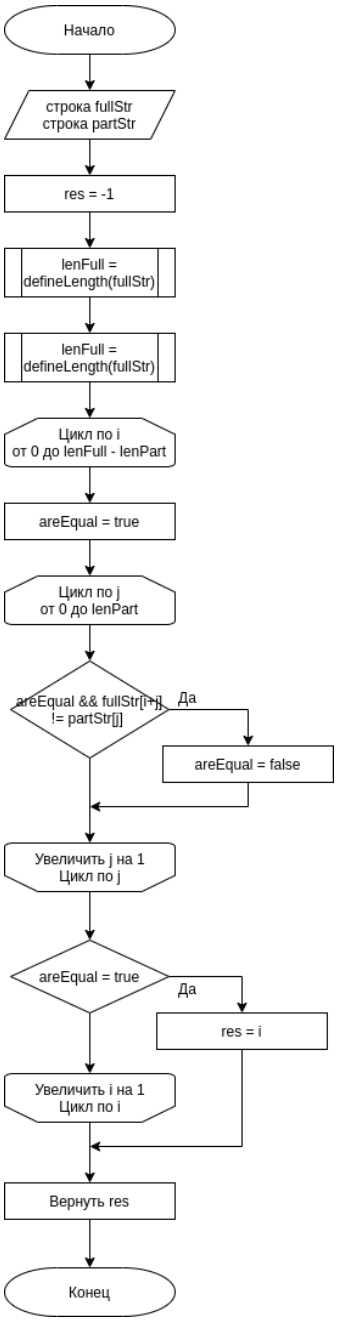

Рисунок 1. Схема стандартного алгоритма.

На рисунке 2 изображена схема *алгоритма Кнута-Морриса-Пратта* поиска подстроки.

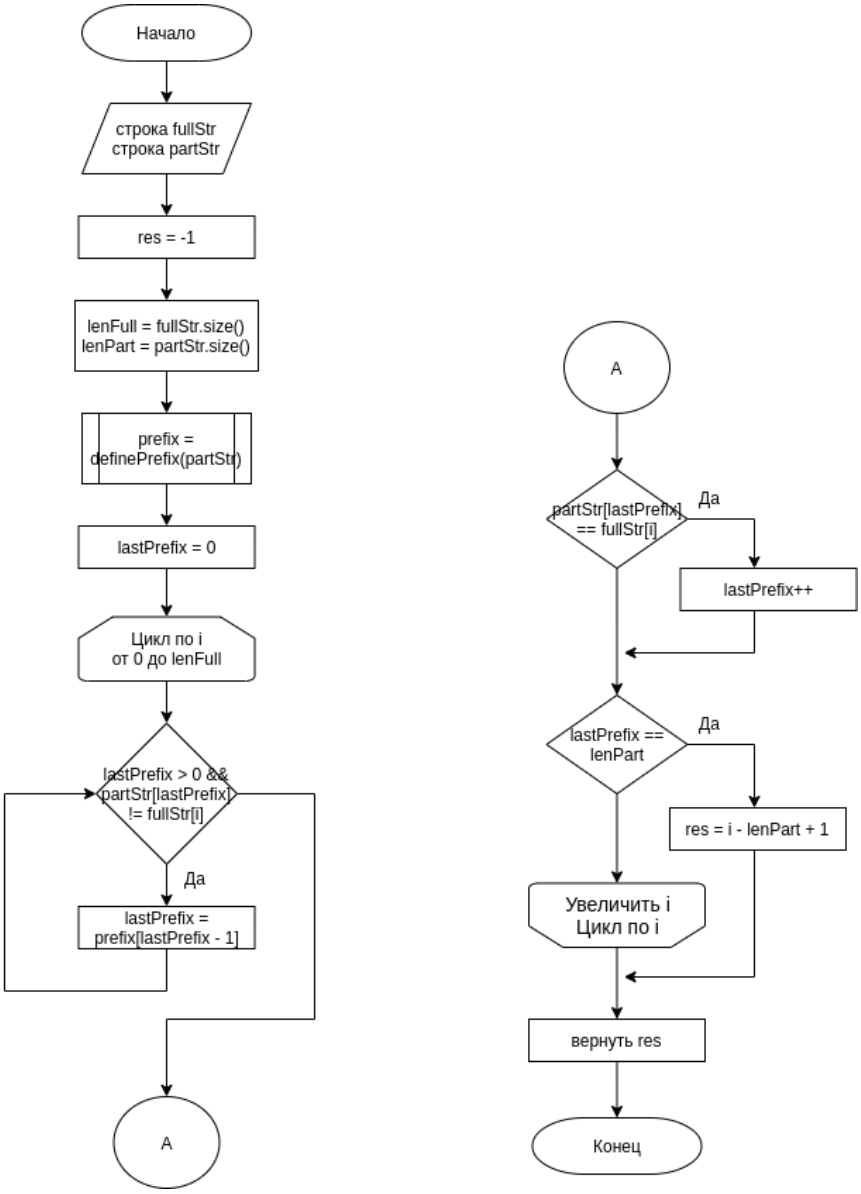

Рисунок 2. Схема алгоритма Кнута-Морриса-Пратта.

На рисунке 3 изображена схема *алгоритма Бойера-Мура* поиска подстроки.

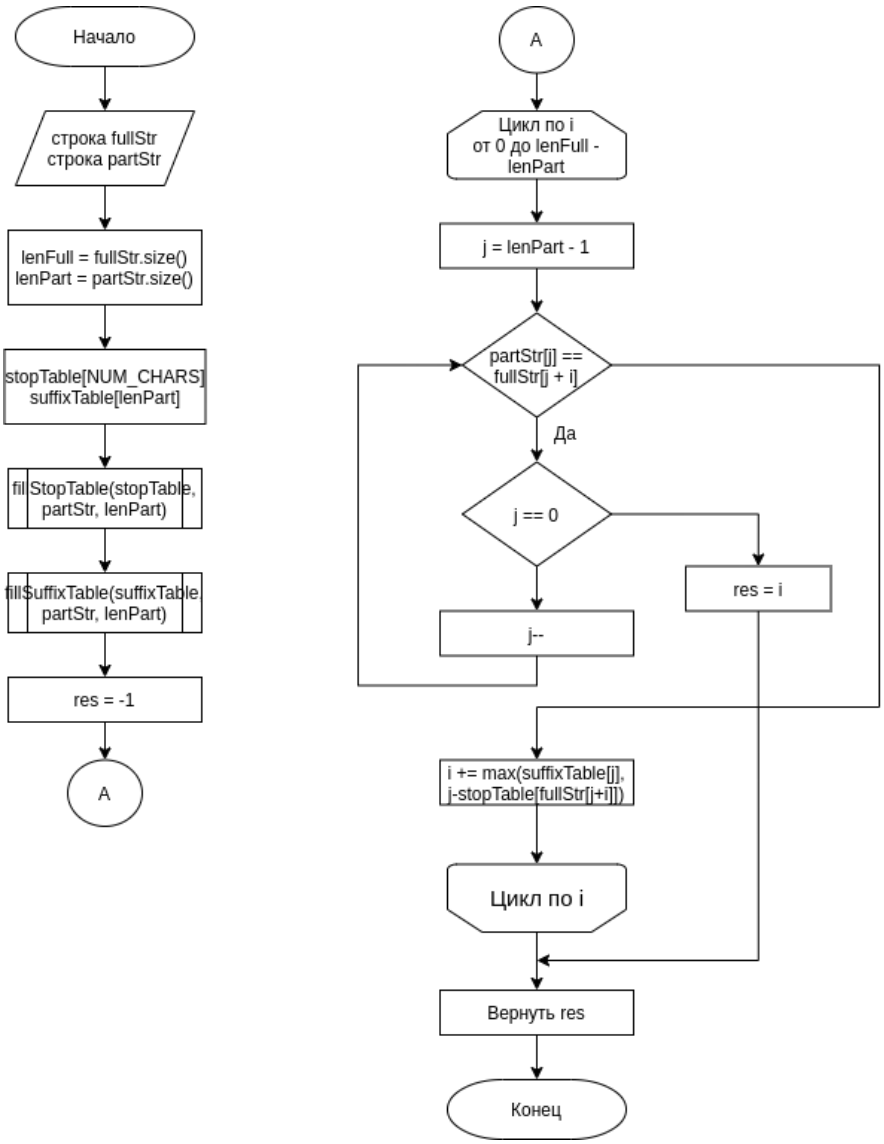

Рисунок 3. Схема алгоритма Бойера-Мура.

2.2. Этапы работы алгоритмов

В данном пункте будет рассмотрено поэтапное выполнение каждого алгоритма.

2.2.1. Стандартный алгоритм

На рисунке 4 представлена пошаговая работа стандартного алгоритма поиска подстроки.

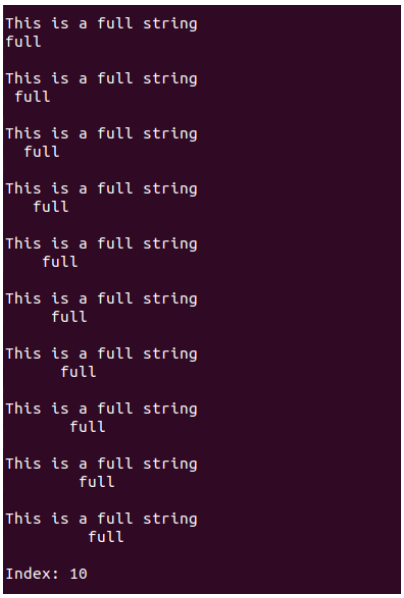

Рисунок 4. Пошаговая работа стандартного алгоритма.

2.2.2. Алгоритм Кнута-Морриса-Пратта

Рассмотрим работу алгоритма для строки “abasabcabcdc” и подстроки “abcabc”.

Массив префиксов имеет вид: [0, 0, 0, 1, 2, 3].

Пошаговая работа алгоритма представлена в таблице 1.

Таблица 1. Пошаговая работа алгоритма Кнута-Морриса-Пратта

a	**b**	**a**	**s**	**a**	**b**	**c**	**a**	**b**	**c**	**d**	**c**
a	b	c	a	b	c						
		a	b	c	a	b	c				
			a	b	c	a	b	c			
				a	b	c	a	b	c		

Результат: 4

a	b	a	s	a	b	c	a	b	c	d	c
				a	b	c	a	b	c		

2.2.3. Алгоритм Бойера-Мура

Рассмотрим работу алгоритма для строки “abcaabcdbcdab” и подстроки “dab”.

Массив стоп-символов имеет вид: [0, 1, 2].

Массив суффиксов имеет вид: [3, 3, 1].

Пошаговая работа алгоритма представлена в таблице 2.

Таблица 2. Пошаговая работа алгоритма Бойера-Мура

a	b	c	a	a	b	c	d	b	c	d	a	b
d	a	b										
			d	a	b							
						d	a	b				
									d	a	b	
										d	a	b

Результат: 10

a	b	c	a	a	b	c	d	b	c	d	a	b
										d	a	b

2.3. Вывод

В данном разделе были рассмотрены 3 схемы алгоритма поиска подстроки в строке: стандартный, Кнута-Морриса-Пратта, Бойера-Мура и поэтапное выполнение каждого.

3. Технологическая часть

В данном разделе будет рассмотрен язык программирования, среда разработки, требуемые инструменты для реализации. Также будет представлена реализация алгоритмов.

3.1. Требования к программному обеспечению

I. Программа должна предусматривать ввод двух слов произвольной длины.
II. Выбор применяемого алгоритма осуществляется пользователем из списка алгоритмов, предложенных в меню.
III. На выходе программа выводит индекс начала подстроки.
IV. Также необходимо предусмотреть выполнение замеров процессорного времени для каждой из реализаций.

3.2. Средства реализации

В данной работе используется язык программирования C++, из-за удобства хранения строк, и опыта написания на нем. Среда разработки – Qt.

Для замеров процессорного времени использовалась функция clock(). [5]

3.3. Реализация алгоритмов

Реализация стандартного алгоритма поиска представлена в листинге 1.

Листинг 1. Алгоритм стандартного поиска.

```
#include "standartSearch.h"

int standartSearch(const string fullStr, const string partStr)
{
    int res = -1;

    int lenFull = defineLength(fullStr);
    int lenPart = defineLength(partStr);

    for (int i = 0; i <= lenFull - lenPart; i++) {
        bool areEqual = true;

        for (int j = 0; j < lenPart && areEqual; j++) {
            if (fullStr[i + j] != partStr[j]) {
                areEqual = false;
            }
        }
```

```
        if (areEqual == true) {
            res = i;
            break;
        }
    }

    return res;
}

int defineLength(const string str)
{
    int len = 0;
    for (len = 0; str[len]; len++);
    return len;
}
```

Реализация алгоритма Кнута-Морриса-Пратта представлена в листинге 2.

Листинг 2. Алгоритм Кнута-Морриса-Пратта.

```
#include "kmpSearch.h"

int kmpSearch(const string fullStr, const string partStr)
{
    int res = -1;

    int lenFull = static_cast<int>(fullStr.size());
    int lenPart = static_cast<int>(partStr.size());

    int* prefix = definePrefix(partStr);
    int lastPrefix = 0;

    for (int i = 0; i < lenFull; i++) {
        while (lastPrefix > 0 && partStr[lastPrefix] != fullStr[i]) {
            lastPrefix = prefix[lastPrefix - 1];
        }

        if (partStr[lastPrefix] == fullStr[i]) {
            lastPrefix++;
        }

        if (lastPrefix == lenPart) {
            res = i - lenPart + 1;
            break;
        }
    }

    delete [] prefix;
    return res;
}

int* definePrefix(const string str)
{
    int len = static_cast<int>(str.size());

    int* prefix = new int[len];
    prefix[0] = 0;
```

```
    int lastPrefix = 0;

    for (int i = 1; i < len; i++) {
        while (lastPrefix > 0 && str[lastPrefix] != str[i]) {
            lastPrefix = prefix[lastPrefix - 1];
        }

        if (str[lastPrefix] == str[i]) {
            lastPrefix++;
        }

        prefix[i] = lastPrefix;
    }

    return prefix;
}
```

Реализация алгоритма Бойера-Мура представлена в листинге 3.

Листинг 3. Алгоритм Бойера-Мура.

```
#include "bmSearch.h"
#include <string.h>

const int NUM_OF_CHARS = 256;

int bmSearch(const string fullStr, const string partStr)
{
    int lenFull = static_cast<int>(fullStr.size());
    int lenPart = static_cast<int>(partStr.size());

    int stopTable[NUM_OF_CHARS + 1];
    fillStopTable(stopTable, partStr, lenPart);

    int suffixTable[lenPart];
    fillSuffixTable(suffixTable, partStr, lenPart);

    int res = -1;

    for (int i = 0; i <= lenFull - lenPart; ) {
        int j = lenPart - 1;

        while (partStr[j] == fullStr[j + i]) {
            if (j == 0) {
                res = i;
                break;
            }
            j--;
        }

        i += max(suffixTable[j], j - stopTable[fullStr[j + i]]);
    }
    return res;
}

void fillStopTable(int stopTable[], const string str, int len)
{
    for (int i = 0; i < NUM_OF_CHARS; i++) {
```

```
        stopTable[i] = -1;
    }

    for (int i = 0; i < len; i++) {
        stopTable[(int)str[i]] = i;
    }
}

void fillSuffixTable(int suffixTable[], const string str, int len)
{
    for (int i = 0; i < len; i++) {
        int offs = len;

        while (offs && !suffixMatch(str.c_str(), len, offs, i)) {
            offs--;
        }

        suffixTable[len - i - 1] = len - offs;
    }
}

int suffixMatch(const char* str, unsigned int len, unsigned int offset, unsigned int suffixlen)
{
    if (offset > suffixlen) {
        return str[offset - suffixlen - 1] != str[len - suffixlen - 1] &&
            memcmp(str + len - suffixlen, str + offset - suffixlen, suffixlen) == 0;
    } else {
        return memcmp(str + len - offset, str, offset) == 0;
    }
}
```

3.4. Описание тестирования

Тестирование осуществляется по принципу «черного ящика».

Для проверки корректности программы необходимо предусмотреть 4 случая расположения искомой подстроки относительно строки: начальное, произвольное, конечное и отсутствие подстроки.

3.5. Вывод

В данном разделе были рассмотрены инструменты, необходимые для реализаций версий алгоритмов поиска подстроки, а также были представлены непосредственно реализации.

4. Экспериментальная часть

В данном разделе будут рассмотрены примеры работы программы, произведено тестирование, выполнены эксперименты по замеру времени, а также выполнен сравнительный анализ полученных данных.

4.1. Примеры работы программы

Взаимодействие с программой (меню, ввод слов, выбор алгоритма, вывод результата) представлено на рисунке 5.

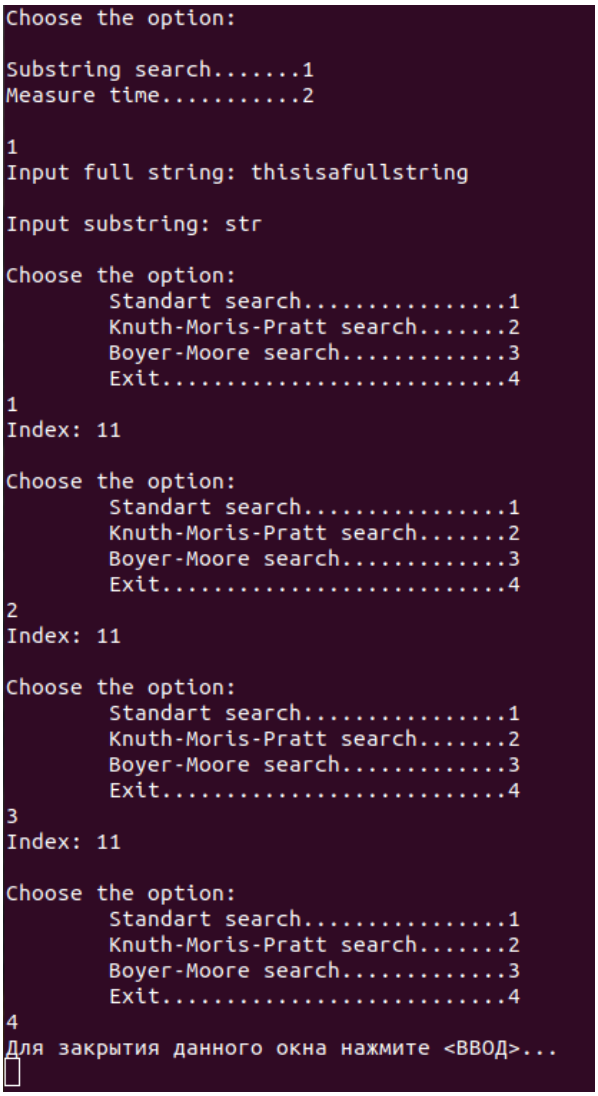

Рисунок 5. Пример взаимодействия с программой.

4.2. Технические характеристики устройства.

Операционная система – LinuxMint 64-bit;

Память – 8 ГБ;

Процессор – Intel™ Core™ i3-7100U CPU @ 2.40 ГГц;

Логических процессов – 4.

4.3. Результаты тестирования

Результаты тестирования приведены в таблице 3.

Таблица 3. Результаты тестирования.

Строка	Подстрока	Ожидаемый результат	Результат
absaabcd	abs	0	0
absaabcd	cd	6	6
absaabcd	aa	3	3
absaabcd	ae	-	“Position not found”

Поскольку результаты работы алгоритмов совпадают, в столбце «Полученный результат» приведено значение, одинаковое для всех реализаций.

Все тесты были пройдены успешно.

4.4. Сравнение реализаций по времени работы

Выполняется серия экспериментов по замеру времени. Каждый эксперимент проводится несколько раз, затем результат делится на их количество для усреднения результатов и сглаживания ошибок. Результаты замеров приведены на рисунке 6.

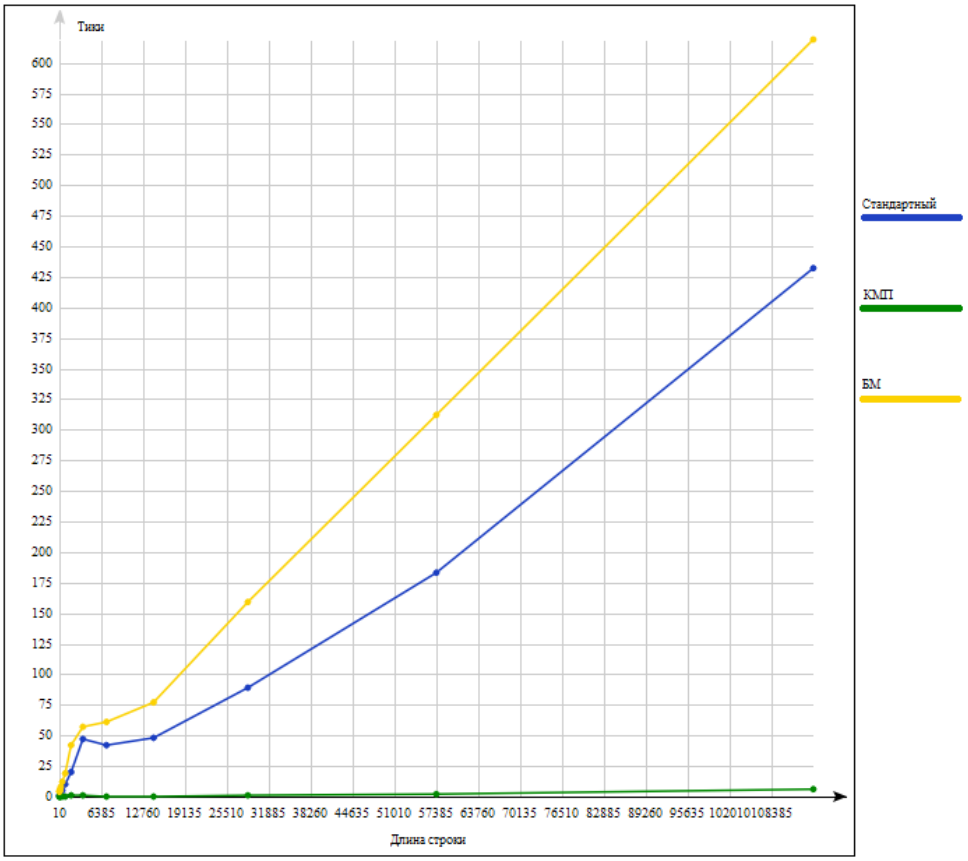

Рисунок 6. График зависимости времени работы алгоритмов от длины строки.

Эксперименты проводились с постоянной длиной искомой подстроки.

Следует отметить, что самым долгим является алгоритм Бойера-Мура. Это связано с тем, что накладные расходы на создание таблицы стоп-слов и таблицы суффиксов превышают выгоду по времени. Самым быстрым алгоритмом является алгоритм Кнута-Морриса-Пратта. График алгоритма является константой. Только на строках большой длины (более 15000 символов) время работы алгоритма немного увеличилось. Что касается стандартного алгоритма, он работает немного быстрее алгоритма Бойера-Мура.

4.5. Вывод

В данном разделе были рассмотрены примеры работы программы, произведено тестирование, выполнены эксперименты по замеру времени, а также выполнен сравнительный анализ полученных данных.

Заключение

Цель работы достигнута, все поставленные задачи выполнены: были разработаны и реализованы три алгоритма поиска подстроки, а также проведен анализ времени работы для каждой реализации.

В результате анализа было установлено, что, как ожидалось, максимальную скорость работы имеет алгоритм Кнута-Морриса-Пратта. Алгоритм стандартного поиска работает медленнее: в 4 раза замедляется время работы на строках до 500 символов длиной. На строках малой длины (до 10 символов) время работы алгоритмов разнится незначительно.

Самым долгим алгоритмом является алгоритм Бойера-Мура, он работает в 2 раза медленнее, чем алгоритм Кнута-Морриса-Пратта, поскольку дополнительные расходы на создание массивов превышают выгоду по времени. Для организации работы алгоритма требуется сформировать два массива: массив стоп-слов и массив суффиксов.

Список литературы

1. Смит Б. Методы и алгоритмы вычислений на строках = Computing Patterns in Strings. — М.: Вильямс, 2006. — 496 с. — ISBN 5-8459-1081-1, 0-201-39839-7.
2. Окулов С. М. Алгоритмы обработки строк. — М.: Бином, 2013. — 255 с. — ISBN 978-5-9963016-2-1.
3. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ = Introduction to Algorithms / Под ред. И. В. Красикова. — 2-е изд. — М.: Вильямс, 2005. — 1296 с. — ISBN 5-8459-0857-4.
4. Гасфилд Д. Строки, деревья и последовательности в алгоритмах: информатика и вычислительная биология = Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology / пер. с англ. И. В. Романовский. — СПб.: Невский Диалект, 2003. — 654 с. — ISBN 5-7940-0103-8.
5. Техническая документация.-URL: https://docs.microsoft.com/ru-ru/cpp/c-runtime-library/reference/clock?view=vs-2019 (дата обращения: 20.09.2020). Текст: электронный.