Государственное образовательное учреждение высшего профессионального образования
“Московский государственный технический университет имени Н.Э.Баумана”

Дисциплина: Анализ алгоритмов

Лабораторная работа №7

Поиск подстроки в строке

Студент группы ИУ7-55Б,
Руднев К. К.,

Преподаватель,
Волкова Л. Л.,
Строганов Ю. В.

2019 г.

Оглавление

Введение

Цель работы: изучение алгоритмов поиска подстроки в строке.
Задачи данной лабораторной работы:

1. изучить алгоритмы Бойера-Мура и Кнута-Морриса-Пратта;
2. реализовать эти алгоритмы;
3. провести тестирование ПО.

Глава 1

Аналитическая часть

В рамках раздела будут рассмотрены алгоритмы поиска подстроки в строке.

1.1 Общие сведения об алгоритмах поиска подстроки в строке

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[1].

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.

1.1.1 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[2].

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 сивола в алфавите. Для каждогостречающегося 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[2].

1.1.3 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 отсительно конкретного элемента подстроки. Создание этого массива позволяет при несовпадении строки сдвигать ее на расстояние, большее, чем 1 (в отличие от стандартного алгоритма).

1.2 Вывод

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

Глава 2

Конструкторская часть

В данном разделе будут рассмотрена пошаговая работа алгоритмов.

2.1 Разработка алгоритмов

В таблице 2.1 и 2.2 будет рассмотрена пошаговая работа алгоритмов Кнута-Морриса-Пратта и Бойера-Мура на значениях строки s и подстроки sub.

string s = "ababacabaa";
string sub = "abaa";

2.1.1 Алгоритм Кнута-Морриса-Пратта

Для алгоритма Кнута-Морриса-Пратта вычисленный массив префиксов для заданой подстроки sub имеет значение: prefix = [0, 0, 1, 1]

Таблица 2.1 отображает пошаговую работу алгоритма Кнута-Морриса-Пратта при данном массиве префиксов.

Таблица 2.1: 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

2.1.2 Алгоритм Бойера-Мура

Для алгоритма Бойера-Мура вычисленный массив суффиксов для заданой 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 длиние sub.

Таблица 2.2: Пошаговая работа алгоритма Бойера-Мура

a	b	a	b	a	c	a	b	a	a
a	b	a	a						
		a	b	a	a				

2.2 Вывод

В данном разделе была разобрана работа алгоритмов на конкретных входных данных.

Глава 3

Технологическая часть

В рамках раздела будут описаны инструментарии разработки, выбор среды, требования к ПО. Также будут предоставлены листинги конкретных реализаций алгоритмов.

Замеры времени были произведены на: Intel(R) Core(TM) i7-8565U, 4 ядра, 8 логических процессоров.

3.1 Средства реализации

Для реализации алгоритмов использовался язык программирования C++ 17 и среда разработки QtCreator Community Edition 5.5. У данного языка имеются удобные библиотеки для написания мультипоточных приложений, чего будет достаточно для реализации текущей лабораторной работы, а среда разработки имеет бесплатную коммьюнити версию и подходящий компилятор.

Замер времени реализован с помощью функции clock() из библиотеки time.h. Измеряется время исполнения кода чистого алгоритма (без учета времени на создание потоков, очередей, генерацию данных и т.п.).

3.2 Требования к программному обеспечению

На вход программа должна получать строку и подстроку, которую необходимо найти.

На выход программа должна вернуть индекс начала совпадения или -1, если подстрока не содержится в данной строке.

3.3 Листинг кода

На листингах 3.1-3.3 представлена реализация конвейерной обработки данных. На листинге 3.1 представлен код стандартного алгоритма поиска подстроки в строке. На листинге 3.2 представлен код алгоритма Бойера-Мура. На листинге 3.3 представлен код алгоритма Кнута-Морриса-Пратта.

Листинг 3.1: Стандартный алгоритм

```
int find(string text, string substr)
{
    for (int i = 0; i < text.length(); ++i)
    {
        int j = 0;
        for (j = 0; j < substr.length(); ++j)
        {
            if(text[i+j] != substr[j])
```

```
                break;
            }
            if (j == substr.length())
                return i;
        }
        return -1;
    }
```

Листинг 3.2: Алгоритм Бойера-Мура

```
    bool isPrefix(const std::string &substr, const int &p)
    {
        int j = 0;
        for (int i = p; i < substr.length(); ++i)
        {
            if (substr[i] != substr[j])
                return false;
            j++;
        }
        return true;
    }

    int suffixLength(const std::string &substr, const int &p)
    {
        int len = 0;
        int i = p, j = substr.length() - 1;
        while (i >= 0 && substr[i] == substr[j])
        {
            len++;
            i--;
            j--;
        }
        return len;
    }

    std::vector<int> getSuffix(const std::string &substr)
    {
        int n = substr.length();
        std::vector<int> table(n);
        int lastPrefixPosition = n;

        for (int i = n - 1; i >= 0; --i)
        {
            if (isPrefix(substr, i + 1))
            lastPrefixPosition = i + 1;
            table[n - 1 - i] = lastPrefixPosition - i + n - 1;
        }

        for (int i = 0; i < n - 1; i++)
        {
            int slen = suffixLength(substr, i);
            table[slen] = n - 1 - i + slen;
        }
```

```
    return table;
}

int searchBM(const std::string &text, const std::string &substr)
{
    std::unordered_map<char, int> stopTable;
    int m = substr.length();
    int n = text.length();
    std::vector<int> suffix = getSuffix(substr);
    for (int i = 0; i < m; ++i)
    {
        stopTable[substr[i]] = m - 1 - i;
    }

    for (int i = m - 1; i < n;)
    {
        int j = m - 1;

        while (substr[j] == text[i])
        {
            if (j == 0)
                return i;
            i--;
            j--;
        }
        auto stop_symbol = stopTable.find(text[i]);
        int stopAdd = stop_symbol != stopTable.end() ? stop_symbol->second : m;
        i += std::max(suffix[m - j - 1], stopAdd);
    }
    return -1;
}
```

Листинг 3.3: Алгоритм Кнута-Морриса-Пратта

```
string::size_type KMP(const string& S, const string& pattern)
{
    vector<int> pf (pattern.length());

    pf[0] = 0;
    for (int k = 0, i = 1; i < pattern.length(); ++i)
    {
        while ((k > 0) && (pattern[i] != pattern[k]))
        k = pf[k-1];

        if (pattern[i] == pattern[k])
            k++;

        pf[i] = k;
    }

    for (int k = 0, i = 0; i < S.length(); ++i)
    {
```

```
            while ((k > 0) && (pattern[k] != S[i]))
                k = pf[k-1];

            if (pattern[k] == S[i])
                k++;

            if (k == pattern.length())
                return (i - pattern.length() + 1);
        }
        return -1;
    }
```

3.4 Вывод

В рамках раздела были предъявлены требования к программному обеспечению. На основании их были разработаны и представлены конкретные реализации стандартного алгоритма, алгоритма Кнута-Морриса-Пратта и алгоритма Бойера-Мура для нахождения подстроки в данной строке.

Глава 4

Экспериментальная часть

В этом разделе будет проведен сравнительный анализ алгоритмов поиска подстроки в строке. На рисунках 4.1 - 4.3 приведены результаты тестов программы.

4.1 Примеры работы

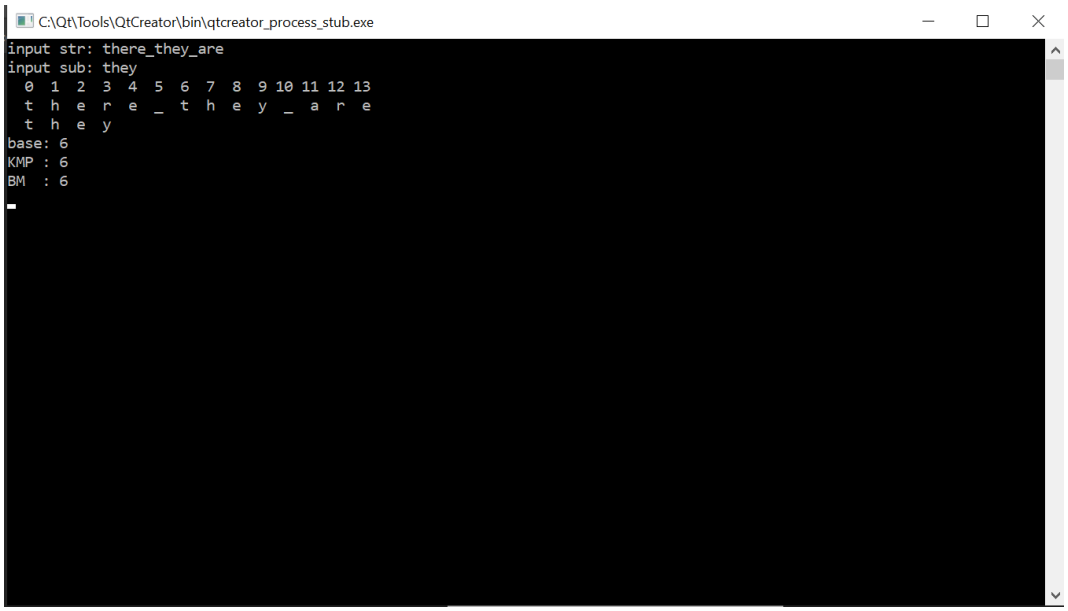

Рис. 4.1: Пример работы 1

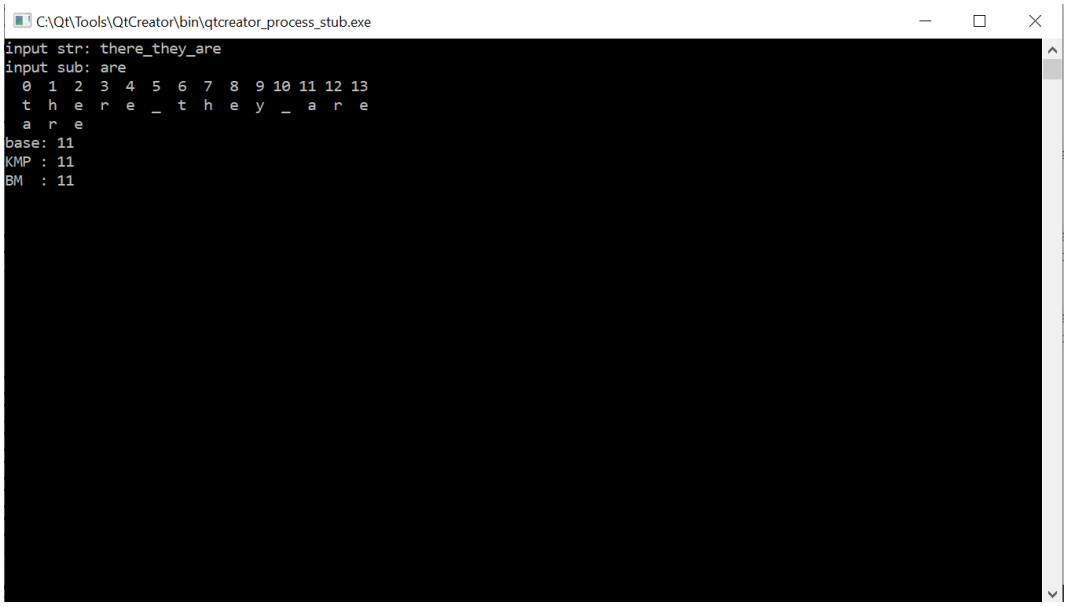

Рис. 4.2: Пример работы 2

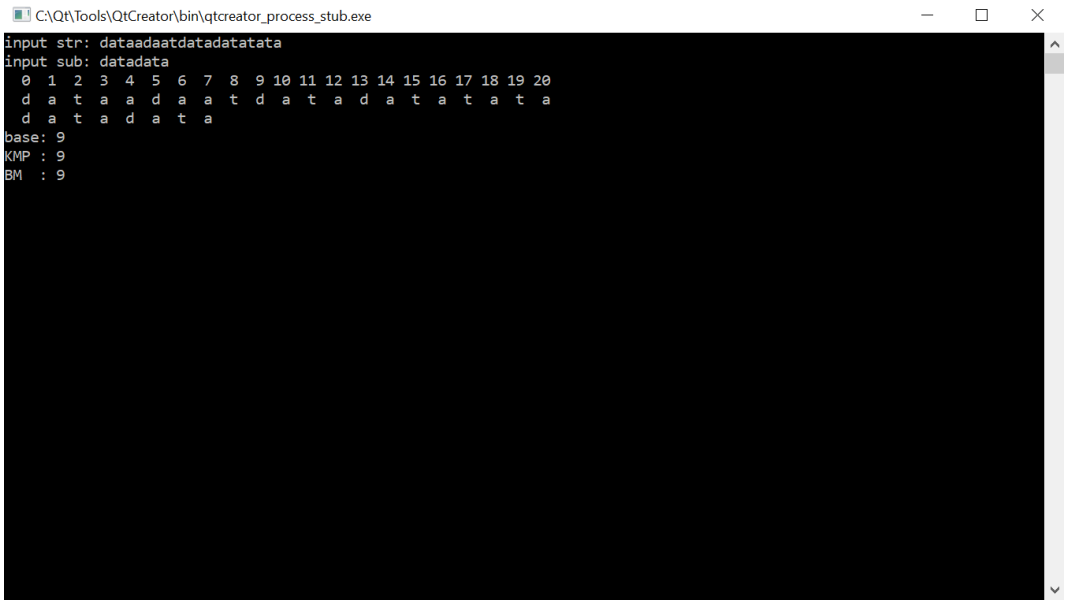

Рис. 4.3: Пример работы 3

4.2 Постановка эксперимента

Были проведены временные эксперименты для строк от 1 000 до 100 000 элементов с шагом 1000. Для каждого замера взят средний результат из 100 замеров. Замеры проведены для трех случаев: длина подстроки 4 символа, длина подстроки 12 символов и длина подстроки 20 символов. Результаты экспериментов представлены на рисунках 4.4-4.6.

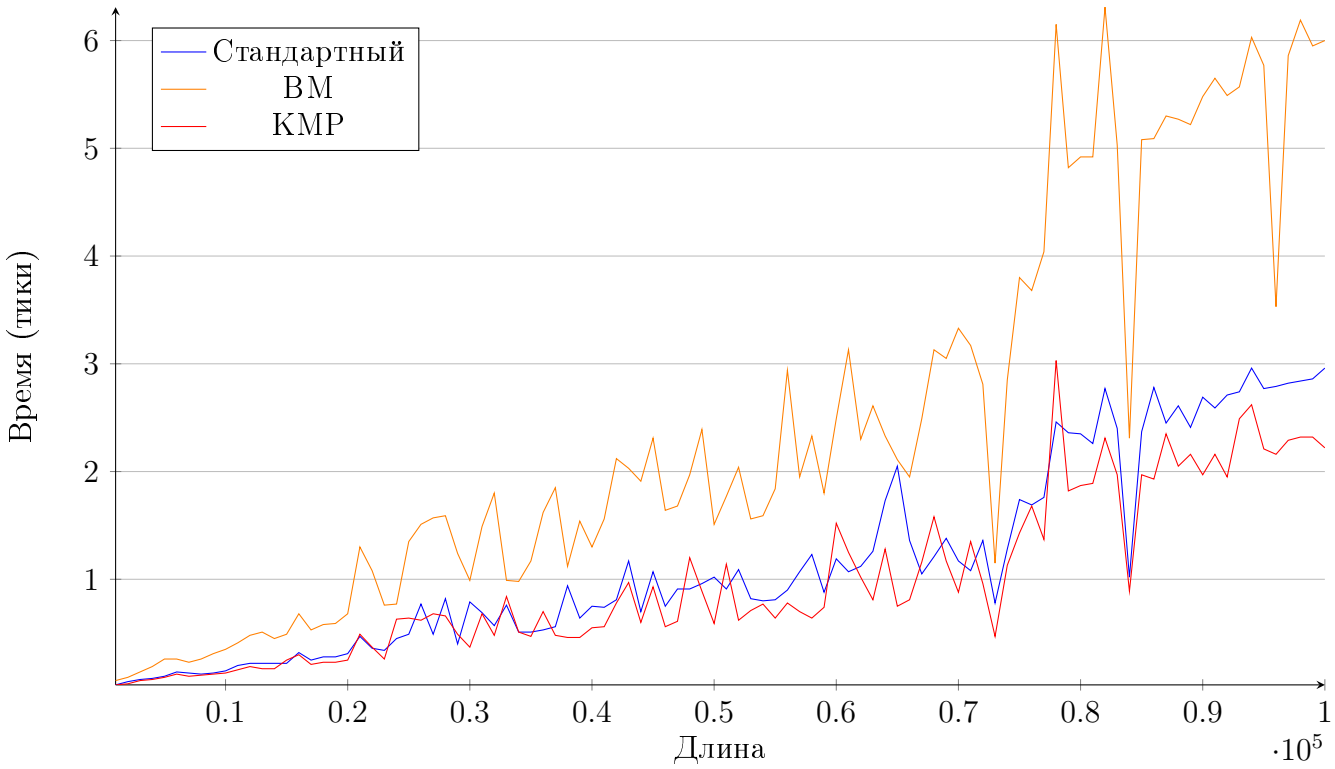

Рис. 4.4: Сравнение времени работы алгоритмов при фиксированной длине подстроки, равной 4

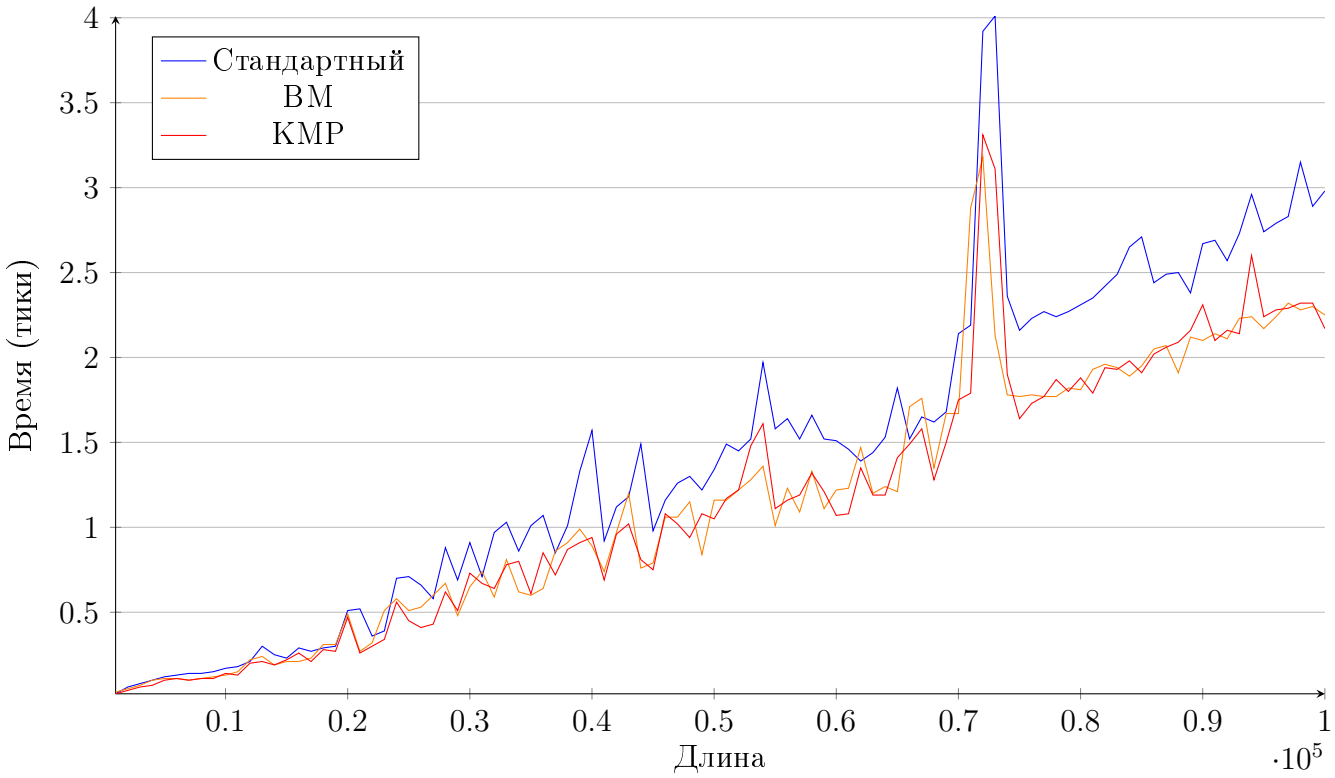

Рис. 4.5: Сравнение времени работы алгоритмов при фиксированной длине подстроки, равной 12

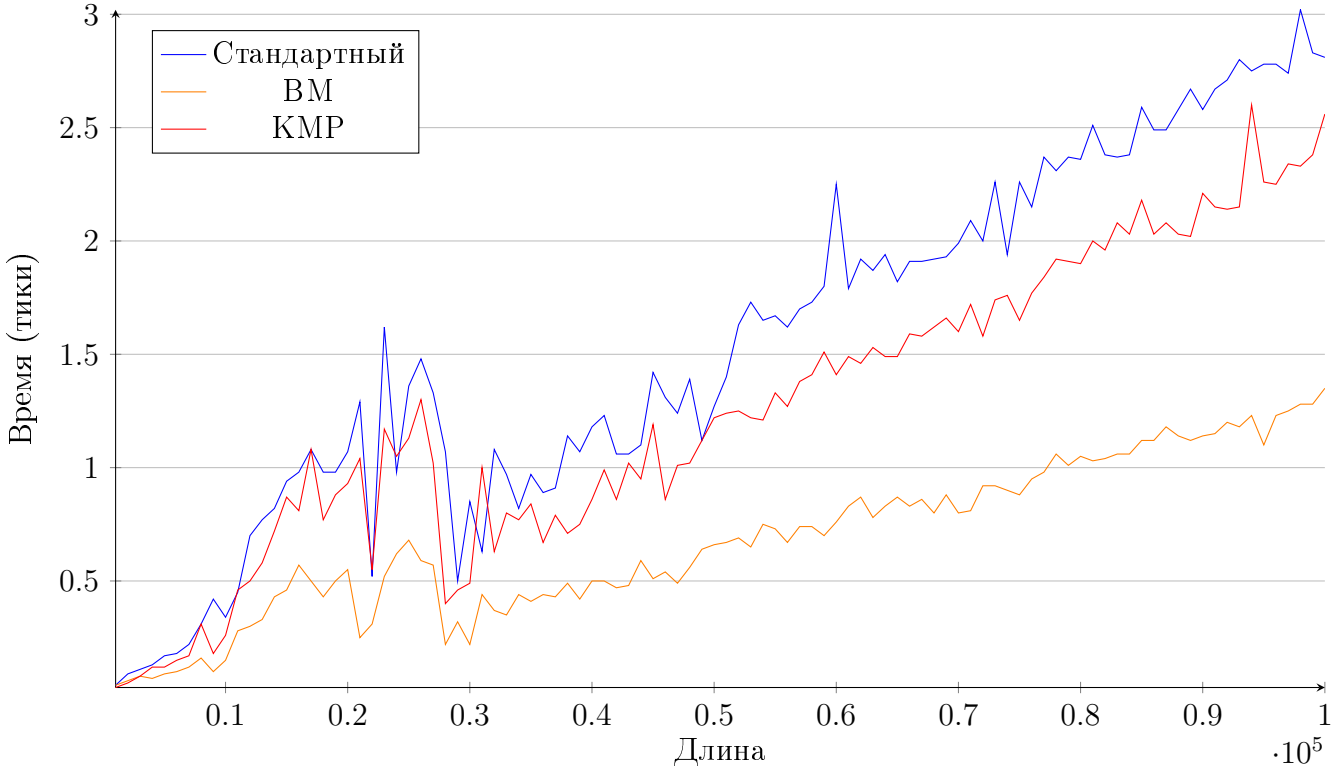

Рис. 4.6: Сравнение времени работы алгоритмов при фиксированной длине подстроки, равной 20

По графикам видно, что время работы алгоритмов сильно зависит от длины подстроки. Алгоритм Кнута-Морриса-Пратта работает лучше стандартного во всех случаях. Алгоритм Бойера-Мура работает лучше всего при длинных подстроках, однако на коротких подстроках этот алгоритм проигрывает даже прямому перебору. Время работы алгоритмов линейно возрастает с увеличением длины строки.

4.3 Вывод

Были проведены эксперименты по замеру времени. Лучшие результаты на длинных подстроках показал алгоритм Бойера-Мура (разница до 80% при длине подстроки в 20 символов). Однако этот алгоритм неприменим для коротких подстрок. Для этого случая лучше применять алгоритм Кнута-Морриса-Пратта (разница до 300% при длине подстроки в 4 символа).

Заключение

В ходе лабораторной работы были изучены и реализованы следующие алгоритмы решения задачи поиска подстроки в строке: стандартный алгоритм, алгоритм Кнута-Морриса-Пратта и алгоритм Бойера-Мура. Был проведен их сравнительный анализ, в ходе которого были получены результаты замеров времени работы алгоритмов в зависимости от длины подстроки. Наиболее стабильным оказался алгоритм Кнута-Морриса-Пратта, однако алгоритм Бойера-Мура показывает лучшие результаты при работе с длинными подстроками.

Литература

[1] Окулов С. М. Алгоритмы обработки строк. — М.: Бином, 2013. — 255 с.

[2] Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход