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 №7

по курсу «Анализ Алгоритмов»

на тему: «Алгоритмы поиска подстроки в строке»

Студент группы ИУ7-56Б _____ (Подпись, дата) Чупахин М. Д. (Фамилия И.О.)

Преподаватель _____ (Подпись, дата) Волкова Л. Л. (Фамилия И.О.)

Москва — 2023 г.

Содержание

Введение

В современном мире, где объемы данных неуклонно растут, эффективные по времени алгоритмы обработки строк являются ключевым элементом многих приложений и систем. Одним из важных задач в области строковых операций является поиск подстроки в строке. Эта проблема возникает в различных контекстах, начиная от поиска информации в текстовых документах и заканчивая обработкой данных в базах данных [1].

Главной целью данной лабораторной работы является исследование алгоритмов поиска подстроки в строке.

Ниже приведен набор задач, выполнение которого необходимо для достижения поставленной цели.

1) Привести описание двух алгоритмов поиска подстроки в строке (стандартный алгоритм и алгоритм Бойера — Мура).

2) Разработать программное обеспечение, реализующее следующие алгоритмы:

   — стандартный алгоритм;

   — алгоритм Бойера — Мура;

3) Сравнить эффективность по времени рассматриваемых алгоритмов.

1 Аналитическая часть

В данном разделе будут рассмотрены два метода поиска подстроки в строке: стандартный алгоритм и алгоритм Бойера — Мура.

1.1 Стандартный алгоритм

Стандартный метод поиска подстроки в строке – это метод «перебора» или «поиска вхождения». Этот метод является простым в реализации, но его эффективность по времени может быть ограничена при работе с большими объемами данных. Суть метода заключается в том, чтобы последовательно проверять каждую позицию в строке на предмет совпадения с искомой подстрокой [2].

Алгоритм поиска вхождения можно описать следующим образом:

1) начать сравнение искомой подстроки с каждым символом в строке, начиная с первого;

2) если символы совпадают, перейти к следующему символу в обеих строках;

3) если символы не совпадают, сдвинуть позицию начала сравнения в строке вправо на один символ и повторить шаг 2;

4) повторять процесс до тех пор, пока не будет найдено точное вхождение подстроки в строку или не будет достигнут конец строки.

Этот метод прост в понимании и реализации, но его основной недостаток заключается в том, что он может иметь квадратичную сложность времени в худшем случае. Например, если искомая подстрока имеет длину m, а строка имеет длину n, то сложность поиска может достигнуть O(m * n) [1].

1.2 Алгоритм Бойера — Мура

Алгоритм Бойера — Мура отличается от метода «перебора» более сложной стратегией сдвига при несовпадении символов. Этот алгоритм минимизирует количество сравнений и основан на двух ключевых идеях: правиле «хорошего суффикса» и правиле «плохого символа» [2].

Основные шаги алгоритма Бойера — Мура:

1) предобработка паттерна (искомой подстроки):
   - составить таблицу сдвигов для правила хорошего суффикса, которая определяет, куда следует сдвигаться при несовпадении с суффиксом паттерна;
   - составить таблицу сдвигов для правила плохого символа, которая определяет, куда следует сдвигаться при несовпадении с символом в паттерне;
2) поиск в строке:
   - начать поиск с конца паттерна;
   - при сравнении символа паттерна с символом строки:
     - если символы совпадают, продолжить сравнивать в обратном направлении;
     - при несовпадении применить правило хорошего суффикса и правило плохого символа для определения оптимального сдвига.

Алгоритм Бойера — Мура эффективен по времени в среднем и в худшем случае. Его преимущества проявляются особенно в случаях, когда в строке и подстроке много повторяющихся символов или суффиксов [1].

Вывод

В данном разделе были рассмотрены два алгоритма поиска подстроки в строке: стандартный алгоритм и Бойера — Мура.

2 Конструкторская часть

2.1 Введение

В данном разделе представлены схемы алгоритмов поиска подстроки в строке: стандартного алгоритма и алгоритма Бойера — Мура.

2.2 Требования к программному обеспечению

Программное обеспечение должно удовлетворять следующим функциональным требованиям: на входе – две строки, на выходе – массив индексов исходной строки.

Программное обеспечение также должно соответствовать следующим требованиям:

— наличие пользовательского интерфейса для выбора действий;

— составление массива индексов вхождения подстроки в строку;

— предоставление функционала для измерения времени выполнения алгоритмов.

2.3 Разработка алгоритмов

В данном разделе представлены схемы алгоритмов для стандартного алгоритма поиска (рисунок 2.1) и алгоритма Бойера — Мура поиска подстроки в строке (рисункок 2.2).

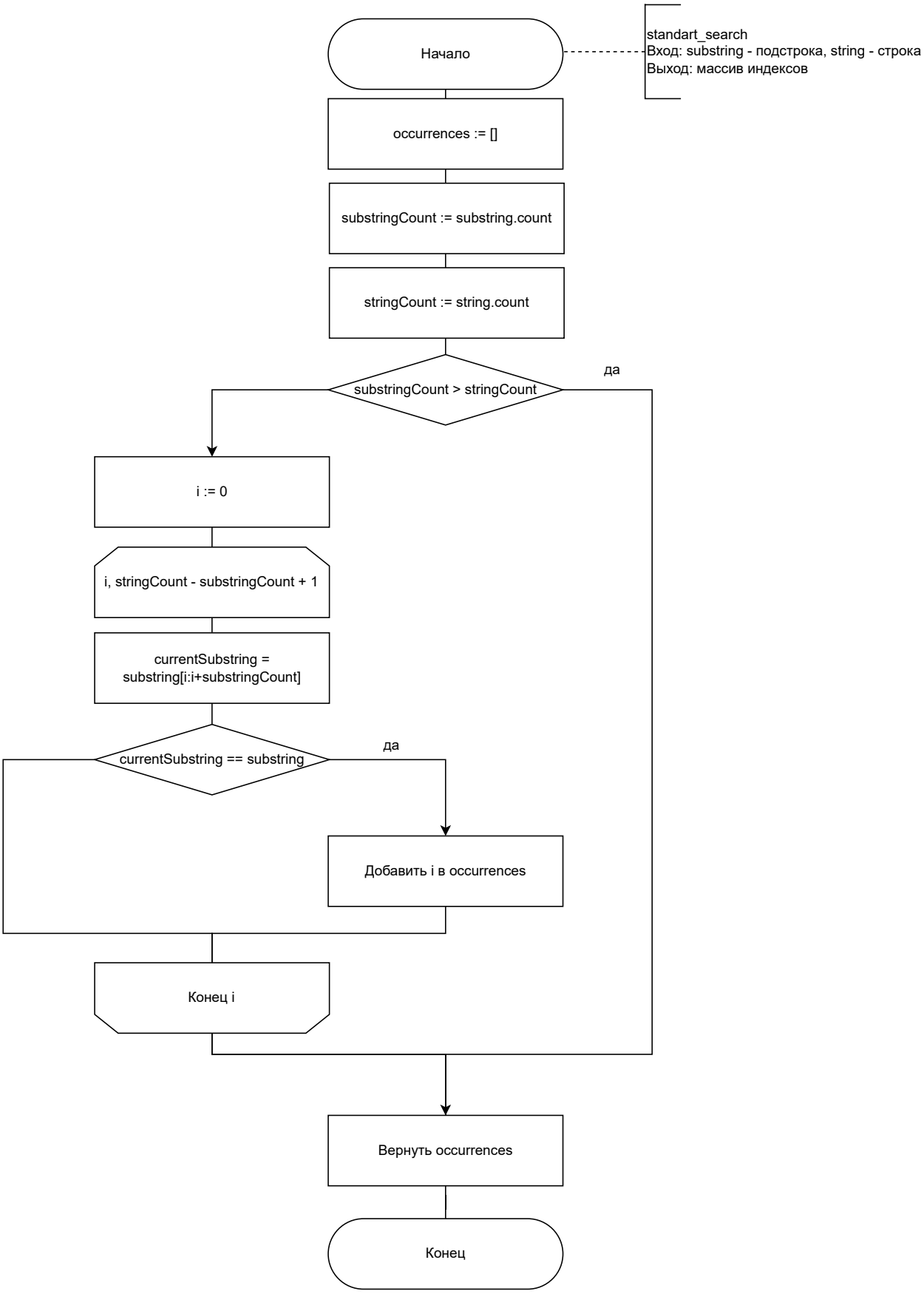

Рисунок 2.1 – Схема стандартного алгоритма поиска подстроки в строке

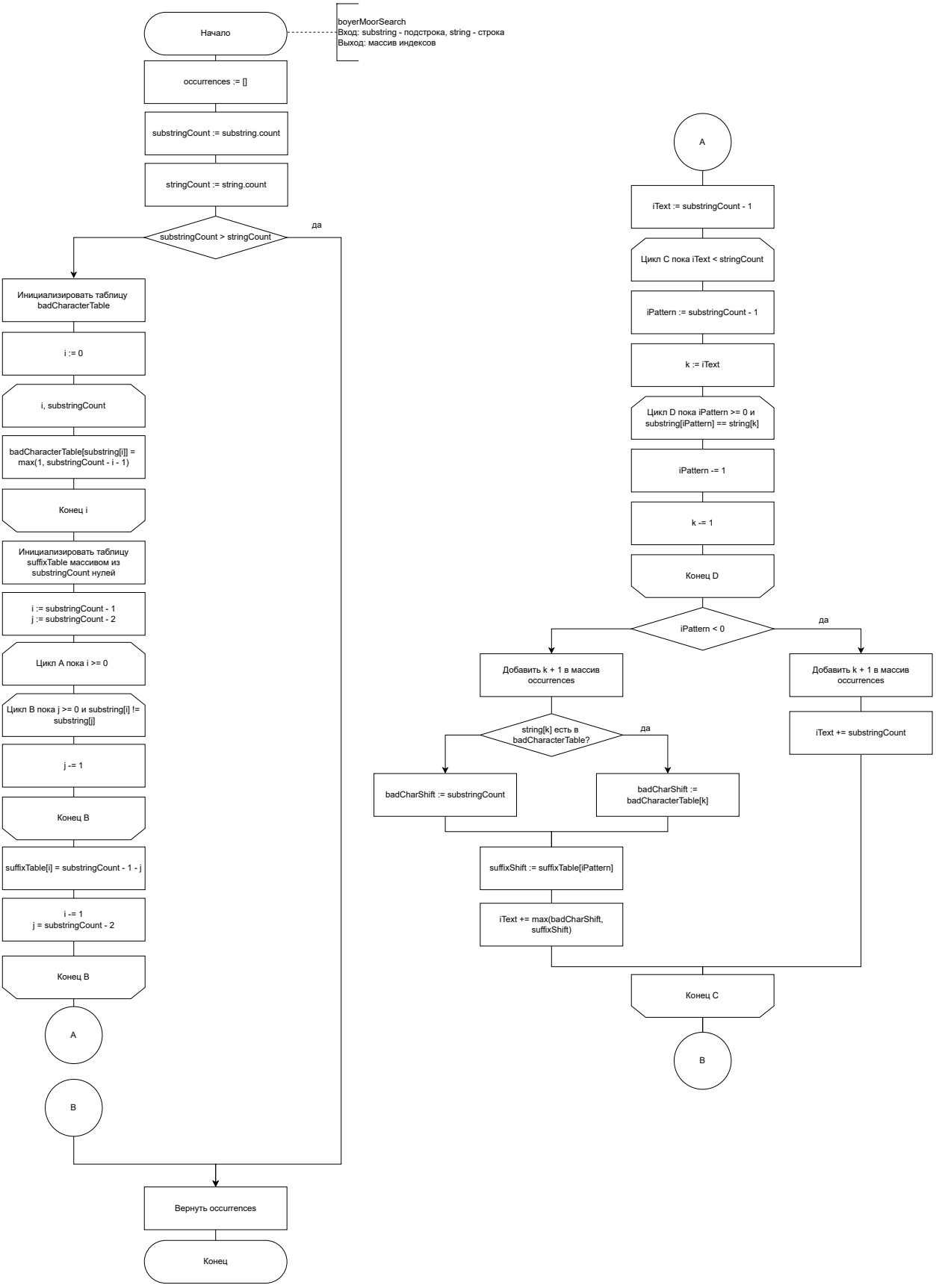

Рисунок 2.2 – Схема алгоритма Бойера — Мура

Вывод

В данном разделе были представлены схемы алгоритмов поиска подстроки в строке: стандартного алгоритма и алгоритма Бойера — Мура.

3 Технологическая часть

В данной главе описаны средства реализации, приведены листинги кода и функциональные тесты.

3.1 Средства реализации

Для разработки данной лабораторной работы был выбран язык программирования Swift [3]. Этот выбор обусловлен возможностью измерения процессорного времени [4] и соответствием с выдвинутыми техническими требованиями, описанными в пункте 2.2.

Измерение времени выполнения алгоритмов производится с использованием функции *clock_gettime()* [4].

3.2 Сведения о модулях программы

Программа разбита на следующие модули:

— `main.swift` — точка входа в программу, где происходит вызов алгоритмов через интерфейс;

— `Algorithms.swift` — содержит реализации алгоритмов поиска подстроки в строке (стандартного алгоритма и алгоритма Бойера — Мура);

— `CPUTimeMeasure.swift` — измеряет время работы алгоритмов с учетом заданного количества повторений;

— `GraphRenderer.swift` — строит графики для каждого из алгоритмов с учетом заданного количества повторений для каждого алгоритма.

3.3 Реализация алгоритмов

В листингах 3.1 – 3.4 приведены реализации алгоритмов поиска подстроки в строке (стандартного алгоритма и алгоритма Бойера — Мура).

Листинг 3.1 – Реализация стандартного алгоритма

```
1     private static func standartSearch(of substring: String, in
         string: String) -> [Int] {
2         var occurrences: [Int] = []
3         let stringArray = Array(string)
4         let stringCount = string.count
5         let substringCount = substring.count
6
7         guard stringCount >= substringCount else {
8             return occurrences
9         }
10
11        for i in 0...(stringCount - substringCount) {
12            let currentSubstring =
                 String(stringArray[i..<(i+substringCount)])
13
14            if currentSubstring == substring {
15                occurrences.append(i)
16            }
17        }
18
19        return occurrences
20    }
```

Листинг 3.2 – Реализация алгоритма Бойера — Мура (Часть 1)

```
1     private static func boyerMooreSearch(of pattern: String, in
         text: String) -> [Int] {
2         let m = pattern.count
3         let n = text.count
4
5         guard m > 0 && n >= m else {
6             return []
7         }
8
9         var resultIndices: [Int] = []
10
11        // Создаем таблицу сдвигов
12        var badCharacterTable: [Character: Int] = [:]
13        for (i, char) in pattern.enumerated() {
14            badCharacterTable[char] = max(1, m - i - 1)
15        }
```

Листинг 3.3 – Реализация алгоритма Бойера — Мура (Часть 2)

```
        // Находим суффикс–префикс массив
        var suffixTable = Array(repeating: 0, count: m)
        var i = m - 1
        var j = m - 2

        while i >= 0 {
            while j >= 0 &&
               pattern[pattern.index(pattern.startIndex,
               offsetBy: i)] !=
                pattern[pattern.index(pattern.startIndex,
                   offsetBy: j)] {
                j -= 1
            }

            suffixTable[i] = m - 1 - j
            i -= 1
            j = m - 2
        }

        // Поиск
        var iText = m - 1

        while iText < n {
            var iPattern = m - 1
            var k = iText

            while iPattern >= 0 &&
               pattern[pattern.index(pattern.startIndex,
               offsetBy: iPattern)] ==
                text[text.index(text.startIndex, offsetBy: k)] {
                iPattern -= 1
                k -= 1
            }

            if iPattern < 0 {
                // Найдено совпадение
                resultIndices.append(k + 1)
                iText += m
            } else {
                // Сдвигаем паттерн
```

Листинг 3.4 – Реализация алгоритма Бойера — Мура (Часть 3)

```
                let badCharShift = badCharacterTable[text[text
                    .index(text.startIndex, offsetBy: k)]] ?? m
                let suffixShift = suffixTable[iPattern]
                iText += max(badCharShift, suffixShift)
            }
        }

        return resultIndices
    }
```

3.4 Функциональные тесты

В таблице приведены функциональные тесты для алгоритмов поиска подстроки в строке. Ожидаемый результат – массив индексов вхождений подстроки в строку. Все тесты были успешно пройдены.

Таблица 3.1 – Тесты алгоритмов поиска подстроки в строке

Подстрока	Строка	Ожидаемый результат
ab	aaaababab	[3, 5, 7]
ab	aaacaaca	[ ]
aaaabab	ab	[ ]
aaaaabbbbb	aaaaabbbbb	[0]

Вывод

В данной главе были описаны средства реализации, приведены листинги кода алгоритмов и функциональные тесты, подтверждающие корректность работы алгоритмов.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени:

— Процессор: Apple M1 Pro [5]

— Оперативная память: 32 ГБайт.

— Операционная система: macOS Ventura 13.5.2. [6]

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

На изображении 4.1 представлена иллюстрация работы разработанного программного продукта. Конкретно, демонстрируются результаты выполнения алгоритмов поиска подстроки в строке

```
Меню:
1. Выполнить поиск подстроки в строке
2. Построить графики
0. Выйти

Выберите команду: 1
Введите строку
abab
Введите подстроку
ab
Выполняется поиск:
Стандартный поиск
[0, 2]
Поиск с использованием алгоритма Бойера-Мура
[0, 2]
```

Рисунок 4.1 – Демонстрация работы программы

4.3 Анализ временных характеристик

В данном разделе представлены результаты экспериментов, в которых измерялось время выполнения поиска подстроки в строке в лучшем случае, худшем случае и на случайных данных, где лучший случай – нахождение подстроки в строке на первой позиции, худший – отстутствие подстроки в строке. Данные результаты представлены в таблицах 4.1 – 4.3.

Таблица 4.1 содержит результаты замеров времени выполнения алгоритмов поиска подстроки в строке для строк, являющихся шестнадцатеричным кодом длинами 2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16} в лучшем случае.

Таблица 4.1 – Результаты замеров времени, лучший случай

	Время, мкс	
Длина строки	**Стандартный**	**Бойера — Мура**
256.0	1616.25	1851.25
1024.0	11799.5	2158.0
4096.0	50445.5	6585.0
16384.0	203296.25	56366.25
65536.0	826367.75	849659.75

На основе данных из таблицы 4.1 был построен график (см. рисунок 4.2).

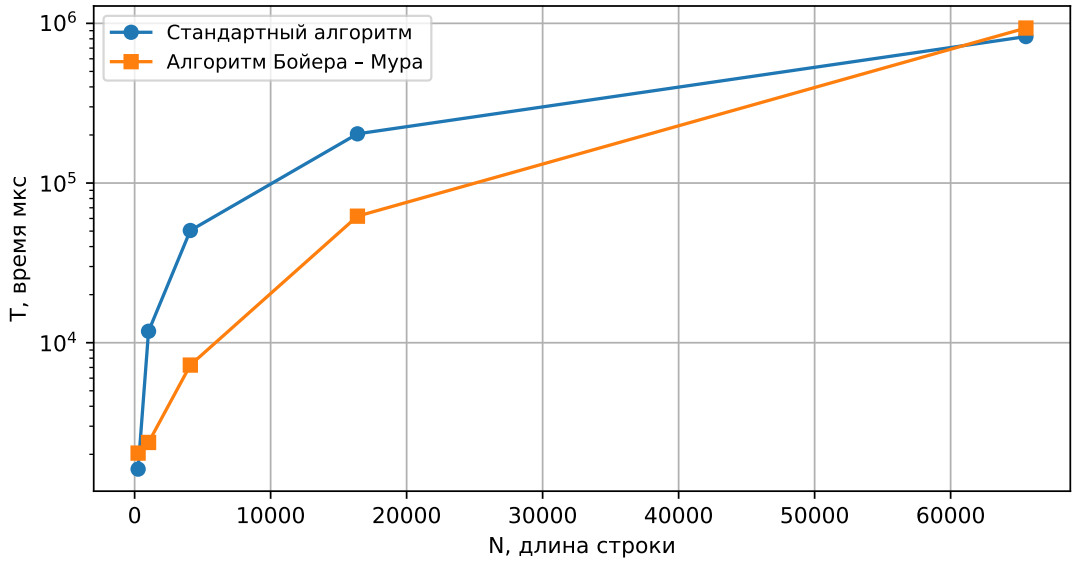

Рисунок 4.2 – Сравнение времени выполнения алгоритмов поиска подстроки в строке, лучший случай

Таблица 4.2 содержит результаты замеров времени выполнения алгоритмов поиска подстроки в строке для строк, являющихся шестнадцатеричным кодом длинами 2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16} в худшем случае.

Таблица 4.2 – Результаты замеров времени, худший случай

Длина строки	**Время, мкс**	
	Стандартный	**Бойера — Мура**
256.0	3501.5	346.75
1024.0	14814.0	228.75
4096.0	51913.75	872.0
16384.0	206123.5	10845.75
65536.0	828979.75	169152.5

На основе данных из таблицы 4.2 был построен график (см. рисунок 4.3).

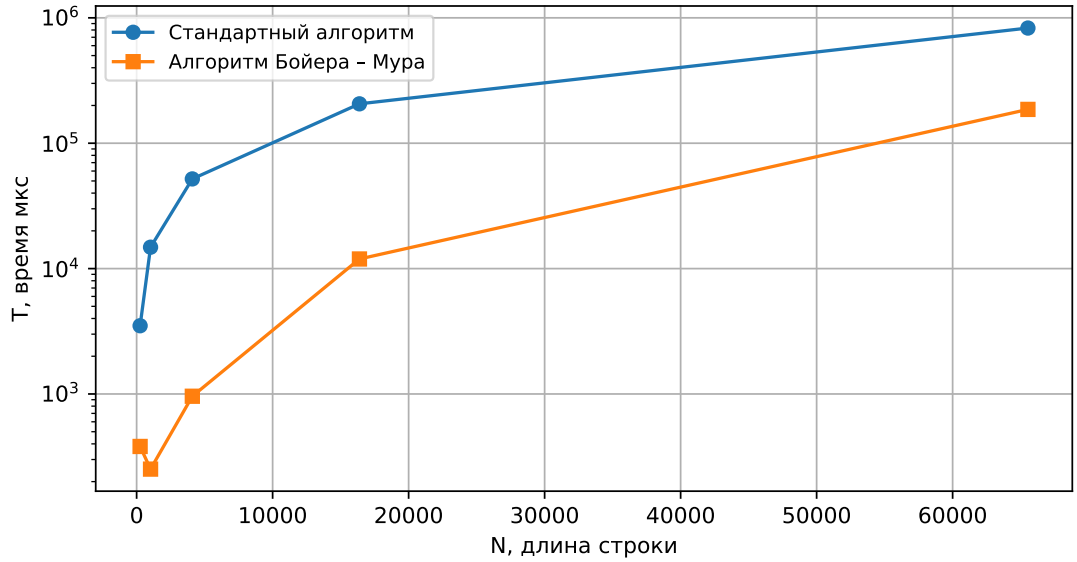

Рисунок 4.3 – Сравнение времени выполнения алгоритмов поиска подстроки в строке, худший случай

Таблица 4.3 содержит результаты замеров времени выполнения алгоритмов поиска подстроки в строке для строк, являющихся шестнадцатеричным кодом длинами 2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16} на случайном наборе данных.

Таблица 4.3 – Результаты замеров времени, случайные данные

Длина строки	**Время, мкс**	
	Стандартный	**Бойера — Мура**
256.0	1739.5	1596.0
1024.0	11682.0	1838.75
4096.0	50740.0	4040.25
16384.0	204447.0	78410.0
65536.0	834890.25	604719.75

На основе данных из таблицы 4.3 был построен график (см. рисунок 4.4).

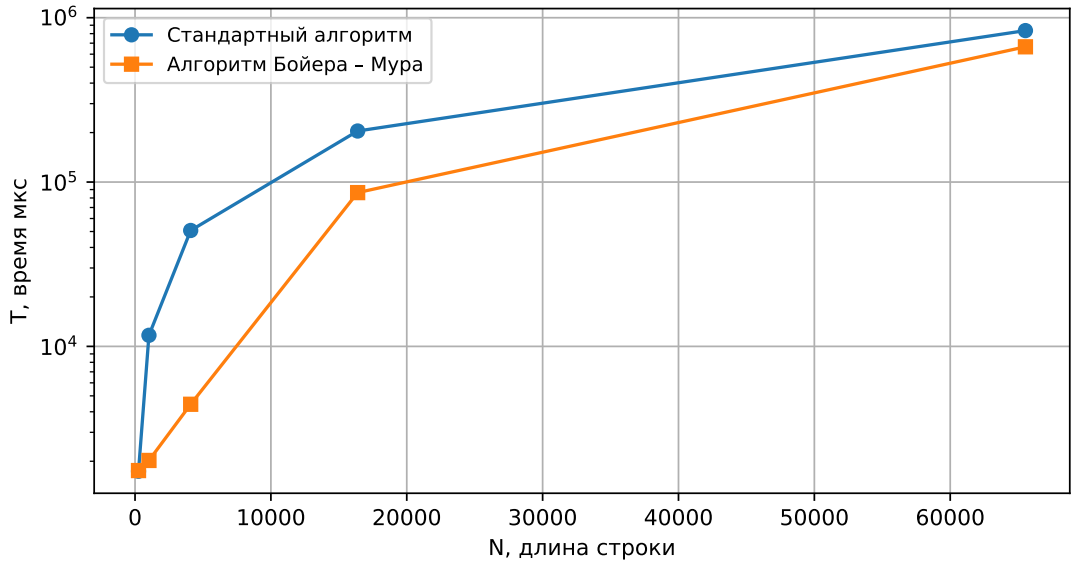

Рисунок 4.4 – Сравнение времени выполнения алгоритмов поиска подстроки в строке, случайный набор данных

4.4 Выводы

Результаты экспериментов позволяют сделать вывод о том, что в лучшем случае реализация алгоритма Бойера – Мура менее эффективна по времени стандартного алгоритма, начиная с размера строки 2^{16}. Наилучшие показатели реализация алгоритма Бойера – Мура показывает в худшем случае – она может быть эффективнее стандартного алгоритма вплоть до 10 раз.

Заключение

По результатам исследований можно утверждать, что в лучшем случае реализация алгоритма Бойера – Мура менее эффективна по времени стандартного алгоритма, начиная с размера строки 2^{16}. Наилучшие показатели реализация алгоритма Бойера – Мура показывает в худшем случае – она может быть эффективнее стандартного алгоритма вплоть до 10 раз.

Основной целью данной лабораторной работы было ислледование алгоритмов поиска подстроки в строке. Ниже представлены выполненные задачи.

1) Приведено описание двух алгоритмов поиска подстроки в строке (стандартный алгоритм и алгоритм Бойера — Мура).

2) Разработано программное обеспечение, реализующее следующие алгоритмы:

   — стандартный алгоритм;

   — алгоритм Бойера — Мура;

3) Были сравнены эффективности рассматриваемых алгоритмов по времени.

Цели и задачи исследования были успешно достигнуты, и полученные результаты позволяют лучше понять и оценить алгоритмы поиска подстроки в строке.

Список использованных источников

1 Обзор методов информационного поиска [Электронный ресурс]. — Режим доступа: https://cyberleninka.ru/article/n/obzor-metodov-informatsionnogo-poiska/viewer (дата обращения: 18.12.2023).

2 Исследование алгоритмов точного поиска подстроки в строке [Электронный ресурс]. — Режим доступа: https://cyberleninka.ru/article/n/issledovanie-algoritmov-tochnogo-poiska-podstroki-v-stroke (дата обращения: 18.12.2023).

3 Документация к Swift [Электронный ресурс]. — Режим доступа: https://developer.apple.com/documentation/swift (дата обращения: 12.09.2023).

4 Измерение процессорного времени [Электронный ресурс]. — Режим доступа: https://man7.org/linux/man-pages/man3/clock_gettime.3.html (дата обращения: 14.09.2023).

5 Процессоры M1 [Электронный ресурс]. — Режим доступа: https://developer.apple.com/documentation/apple-silicon (дата обращения: 15.09.2023).

6 Операционная система macOS. [Электронный ресурс]. — Режим доступа: https://developer.apple.com/macos/ (дата обращения: 15.09.2023).