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»

Тема Алгоритмы поиска

Студент Пронина Л.Ю.

Группа ИУ7-54Б

Оценка (баллы)

Преподаватель Волкова Л. Л.

Москва — 2023 г.

Содержание

Введение

Поиск подстроки является важной операцией в области обработки текстов и анализа данных. Он широко используется в различных приложениях, включая поисковые системы, обработку естественного языка, биоинформатику и многое другое. Поэтому понимание и применение оптимальных алгоритмов поиска подстроки играют важную роль в разработке эффективных приложений.

Целью данной работы является исследование алгоритма Кнута-Морриса-Пратта и его модификацию с использованием эвристики «плохого» символа. Для достижения поставленной цели необходимо выполнить следующие задачи:

— описать алгоритмы решения задачи поиска подстроки в строке – Кнута-Морриса-Пратта и его модификацию с использованием эвристики «плохого» символа;

— реализовать эти алгоритмы;

— провести сравнительный анализ рассматриваемых алгоритмов по времени и по количеству сравнений;

— подготовить отчет по лабораторной работе.

1 Аналитическая часть

В этом разделе будет представлена информация об алгоритме Кнута-Морриса-Пратта и его модификации с использованием эвристики «плохого» символа.

1.1 Алгоритм Кнута-Морриса-Пратта

Алгоритм Кнута-Морриса-Пратта (КМП) — это эффективный алгоритм для поиска подстроки в строке. Он был разработан Дональдом Кнутом и Валтором Моррисом в 1970-х годах, а затем усовершенствован Джеймсом Праттом.

Основная идея алгоритма КМП заключается в использовании префиксной функции, которая предварительно вычисляется для искомой подстроки. Префиксная функция определяет наибольшую длину суффикса подстроки, являющегося ее префиксом. Это позволяет алгоритму оптимально смещаться по строке в случае несоответствия символов, минимизируя количество сравнений [1].

Процесс работы алгоритма КМП включает следующие шаги:

1) Вычисление префиксной функции для искомой подстроки. Для каждой позиции в подстроке определяется длина наибольшего общего префикса и суффикса. Эти значения записываются в отдельный массив.

2) Поиск подстроки в строке. Алгоритм просматривает символы строки и подстроки поочередно. При сравнении символов, если они не совпадают, используется префиксная функция для оптимального смещения указателей. Если совпадение найдено, алгоритм продолжает сравнивать следующие символы.

Алгоритм имеет линейную сложность и работает за $O(n + m)$, где n — длина строки, а m — длина подстроки.

1.2 Модифицированный алгоритм с использованием эвристики «плохого» символа

Модифицированный алгоритм поиска подстроки с использованием эвристики «плохого» символа является улучшенной версией классического алгоритма Бойера-Мура. Он был разработан для повышения эффективности поиска и сокращения количества сравнений.

Основная идея модифицированного алгоритма заключается в использовании таблицы «плохого» символа. Эта таблица предоставляет информацию о смещении, которое можно сделать при несоответствии символов в процессе поиска. При каждом несовпадении символов, алгоритм консультируется с таблицей «плохого» символа и смещает указатель по тексту на максимально возможное количество позиций, основываясь на информации из таблицы.

Таблица «плохого» символа строится на основе позиций символов в подстроке. Для каждого символа в подстроке вычисляется смещение, которое необходимо выполнить в случае его несоответствия с символом в тексте. Если символ не содержится в подстроке, то смещение равно длине подстроки.

В результате использования эвристики «плохого» символа, алгоритм сокращает количество проверок и сравнений, пропуская часть текста, где не может находиться искомая подстрока. Это позволяет значительно повысить производительность поискового алгоритма.

1.3 Вывод

В данном разделе был рассмотрен алгоритм Кнута-Морриса-Пратта и его модификация с использованием эвристики «плохого» символа.

2 Конструкторская часть

В этом разделе будут представлены схемы алгоритма Кнута-Морриса-Пратта и его модификации с использованием эвристики «плохого» символа

2.1 Разработка алгоритмов

На рисунках 2.2 и 2.1 представлены схемы алгоритма Кнута-Морриса-Пратта и вспомогательной функции соответственно, а на рисунках 2.3 и 2.4 схема модифицированного алгоритма с использованием эвристики «плохого» символа.

Рисунок 2.1 – Схема алгоритма определения наибольшей длины суффикса подстроки

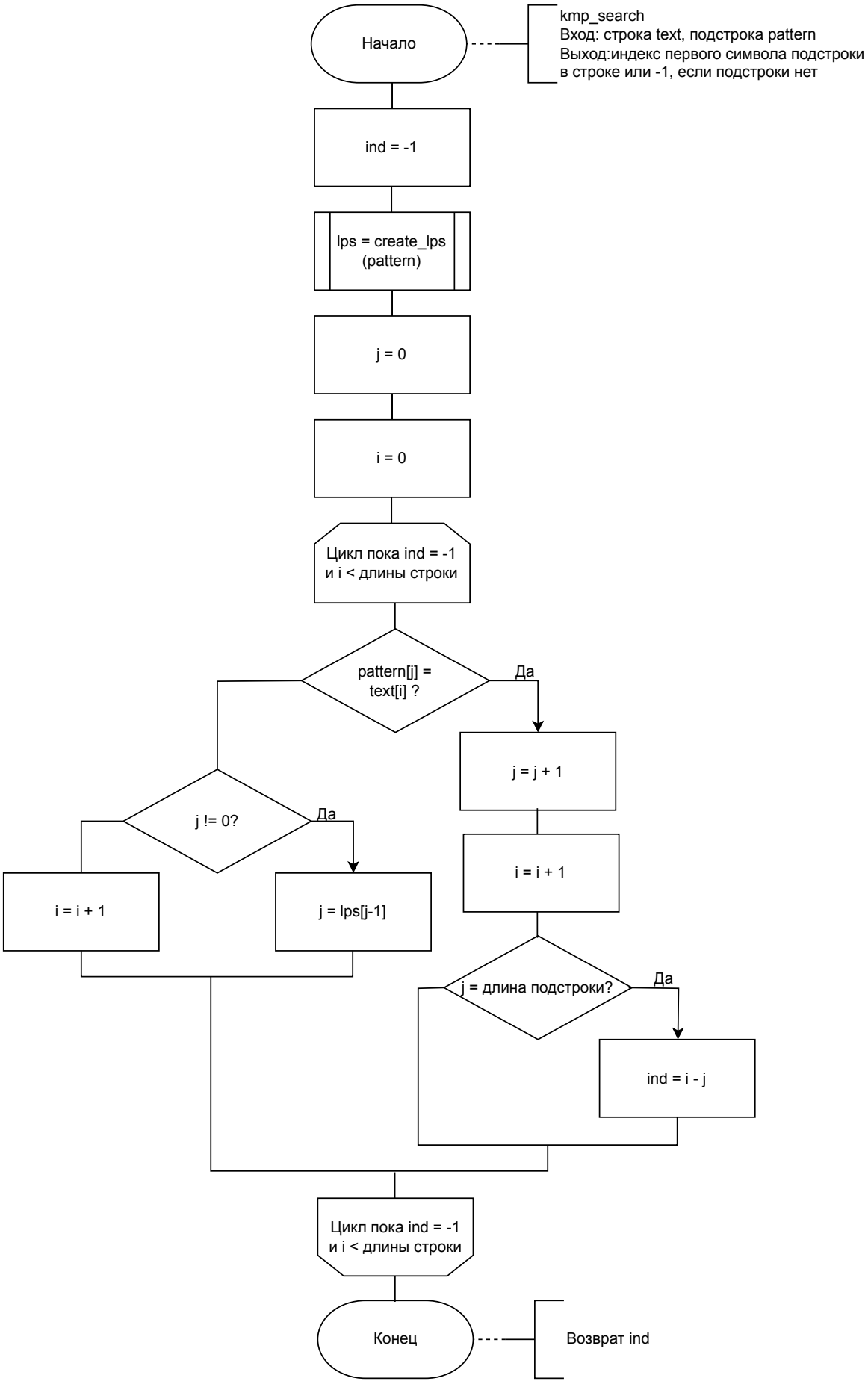

Рисунок 2.2 – Схема алгоритма Кнута-Морриса-Пратта

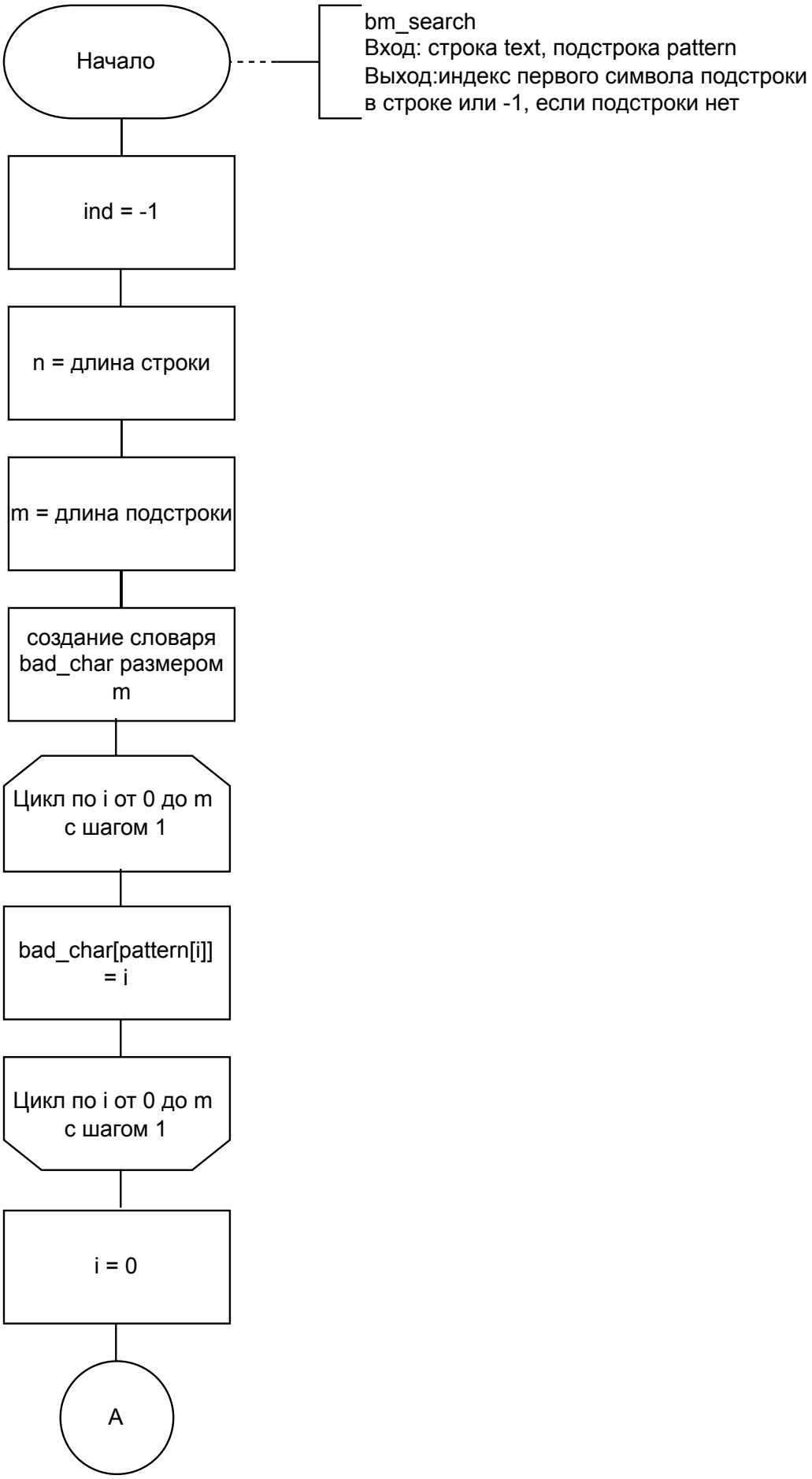

Рисунок 2.3 – Схема модифицированного алгоритма с использованием эвристики «плохого» символа(1 часть)

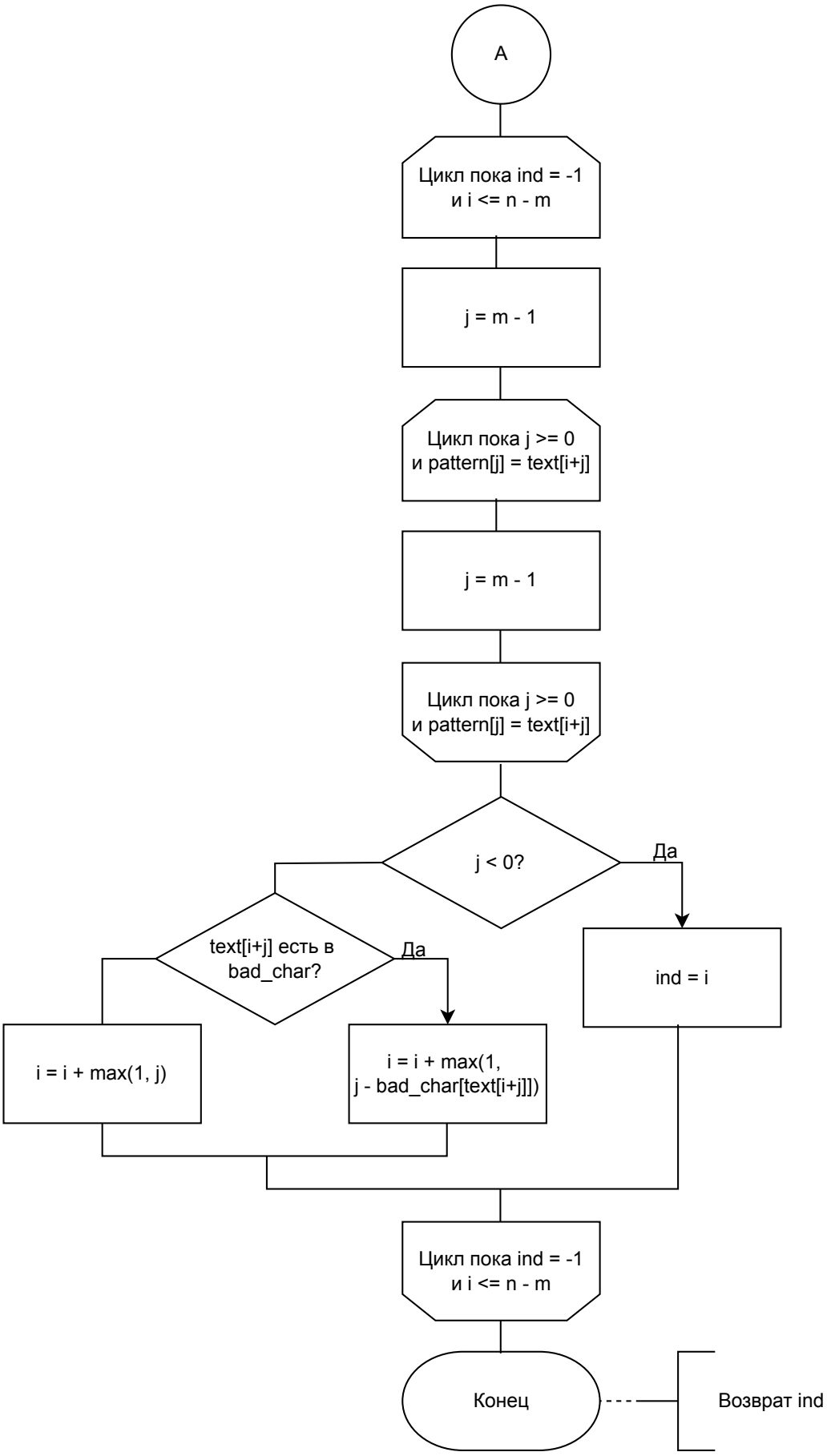

Рисунок 2.4 – Схема модифицированного алгоритма с использованием эвристики «плохого» символа(2 часть)

2.2 Классы эквивалентности при тестировании

Для тестирования выделены классы эквивалентности, представленные ниже.

1) Неверно выбран пункт меню — не число или число, меньшее 0 или большее 5.

2) Неверно введена подстрока — пустая строка.

3) Неверно введена строка — пустая строка.

4) Подстроки нет в строке.

5) Подстрока есть в строке.

2.3 Вывод

В данном разделе были построены схемы алгоритмов, рассматриваемых в лабораторной рабо и были описаны классы эквивалентности для тестирования.

3 Технологическая часть

В данном разделе будут рассмотрены средства реализации, а также представлены листинги алгоритма Кнута-Морриса-Пратта и его модификации с использованием эвристики «плохого» символа.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования *Python*[2]. В текущей лабораторной работе требуется замерить процессорное время для выполняемой программы, а также построить графики. Все эти инструменты присутствуют в выбранном языке программирования.

Время было замерено с помощью функции *process_ time(...)* из библиотеки *time*[3].

3.2 Реализация алгоритмов

В листинге 3.1 представлен алгоритм для определения наибольшей длины суффикса подстроки, являющегося ее префиксом, в листинге 3.2 — алгоритм Кнута-Морриса-Пратта, а в листинге 3.3 его модификация с использованием эвристики «плохого» символа.

Листинг 3.1 – Алгоритм определения наибольшей длины суффикса подстроки

```
def create_lps(pattern):
    m = len(pattern)
    lps = [0] * m
    length = 0
    i = 1
    while i < m:
        if pattern[i] == pattern[length]:
            length += 1
            lps[i] = length
            i += 1
        else:
            if length != 0:
                length = lps[length - 1]
            else:
                lps[i] = 0
                i += 1
return lps
```

Листинг 3.2 – Алгоритм Кнута-Морриса-Пратта

```
def kmp_search(text, pattern):
    n = len(text)
    m = len(pattern)
    lps = create_lps(pattern)
    indices = -1
    i, j = 0, 0
    while i < n:
        if pattern[j] == text[i]:
            i += 1
            j += 1
        if j == m:
            indices = i - j
            break
        else:
            if j != 0:
                j = lps[j - 1]
            else:
                i += 1
return indices
```

Листинг 3.3 – Модифицированный алгоритм с использованием эвристики «плохого» символа

```
def bm_search(text, pattern):
    n = len(text)
    m = len(pattern)
    bad_char = {}
    indices = -1
    for i in range(m):
        bad_char[pattern[i]] = i
    i = 0
    while i <= n - m:
        j = m - 1
        while j >= 0 and pattern[j] == text[i+j]:
            j -= 1
        if j < 0:
            indices = i
            break
        else:
            if text[i+j] in bad_char:
                i += max(1, j - bad_char[text[i+j]])
            else:
                i += max(1, j)
    return indices
```

3.3 Сведения о модулях программы

Программа состоит из двух модулей:

- *main.py* - файл, содержащий меню программы, а также весь служебный код;
- *algorithms.py* - файл, содержащий реализацию рассматриваемых алгоритмов поиска подстроки в строке.

3.4 Функциональные тесты

В таблице 3.1 приведены тесты для функций программы. Тесты *для всех функций* пройдены успешно.

Если функция находит подстроку в строке, то она возвращает индекс первого символа подстроки в строке, иначе она возвращает -1.

Таблица 3.1 – Функциональные тесты

Подстрока	Строка	Полученный индекс	Ожидаемый индекс
test	testik	0	0
est	testik	1	1
one	testikone	6	6
g	testikonesecond	-1	-1

3.5 Вывод

Были представлены листинги всех алгоритмов — Кнута-Морриса-Пратта и его модификации с использованием эвристики «плохого» символа. Также в данном разделе была приведена информации о выбранных средствах реализации и сведения о модулях программы, проведено функциональное тестирование.

4 Исследовательская часть

В данном разделе будет приведен пример работы программы, а также проведен сравнительный анализ алгоритмов при различных ситуациях на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялся эксперимент представлены далее:

1) операционная система — Ubuntu 22.04.3 [4] Linux x86_64;

2) память — 16 Гб;

3) процессор — Intel® Core™ i5-1135G7 @ 2.40 ГГц.

При эксперименте ноутбук не был включен в сеть электропитания.

4.2 Демонстрация работы программы

На рисунке 4.1 представлен результат работы программы для обоих рассматриваемых алгоритмов.

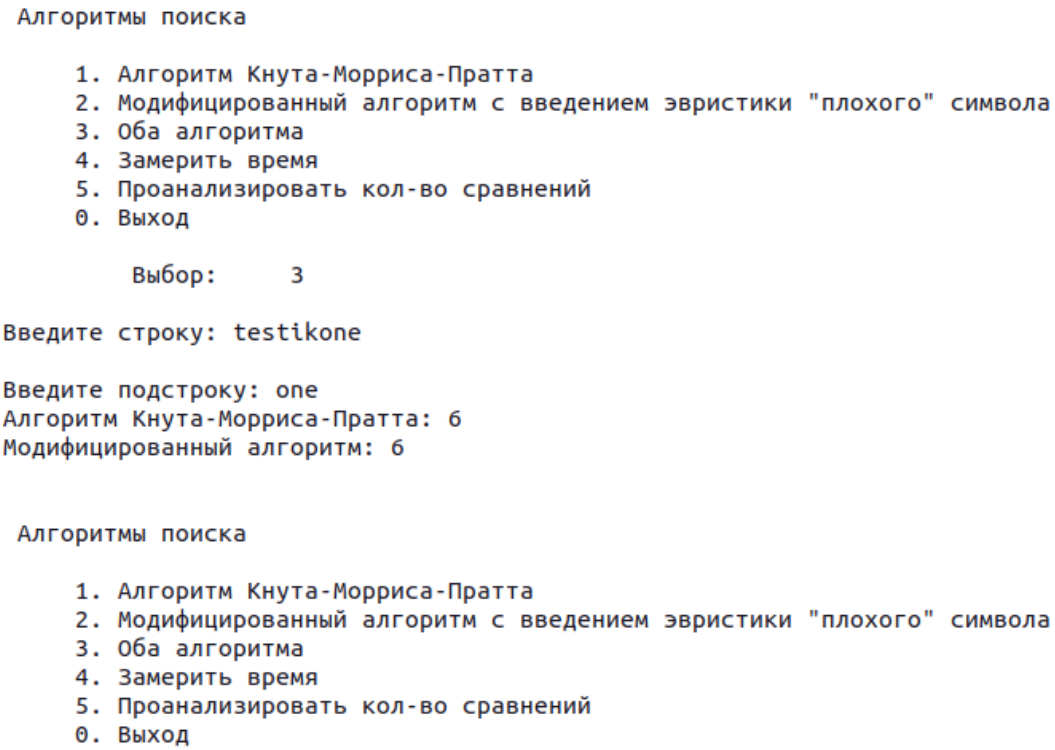

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения алгоритмов

Как было сказано выше, используется функция замера процессорного времени *process_time(...)* из библиотеки *time* на *Python*. Функция возвращает процессорное время типа float в секундах.

Функция используется дважды: перед началом выполнения алгоритма и после завершения, затем из конечного времени вычитается начальное, чтобы получить результат.

Замеры проводились при длине строки в 30 символов, длине подстроки в 3 символа и разном расположении подстроки в строке. Для каждого расположения время находилось 250 раз и затем усреднялось.

Результаты замеров приведены на рисунке 4.2 в графическом виде.

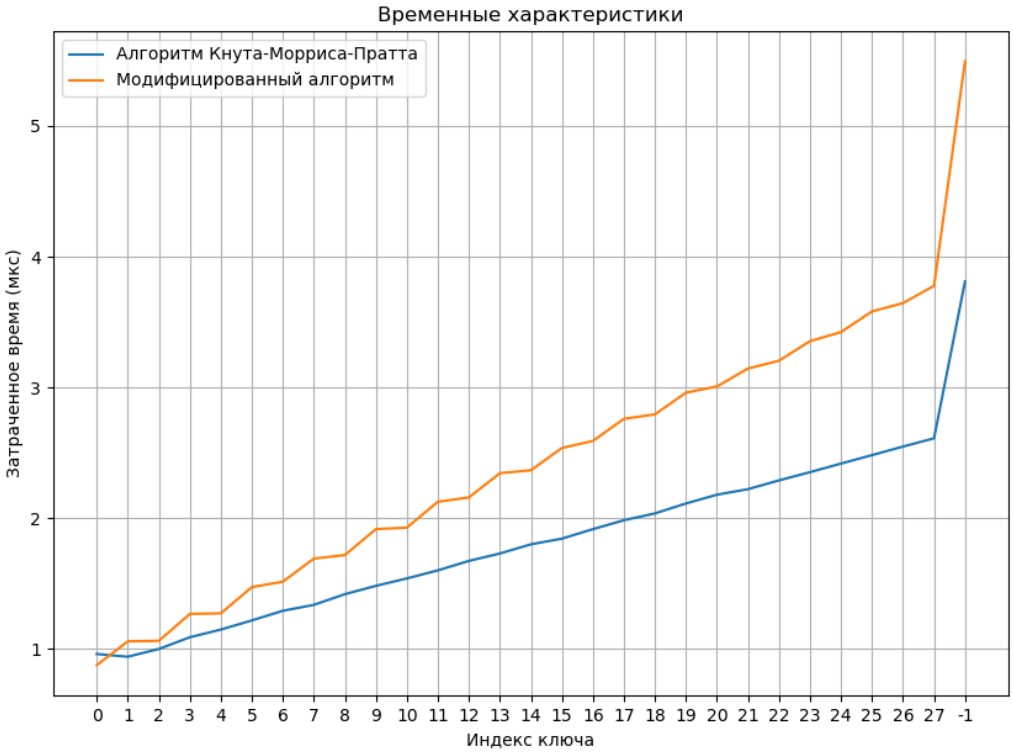

Рисунок 4.2 – Сравнение по времени алгоритмов

4.4 Количество сравнений при работе алгоритмов

Для каждого алгоритма был проведен анализ по количеству сравнений для нахождения подстроки в строке. Строка бралась длиной 33 символа, подстрока длиной 3 символа, расположение — разное.

На рисунке 4.3 представлена гистограмма с количеством сравнений для поиска алгоритмом Кнута-Морриса-Пратта, на рисунке 4.4 — его модификацией с использованием эвристики «плохого» символ.

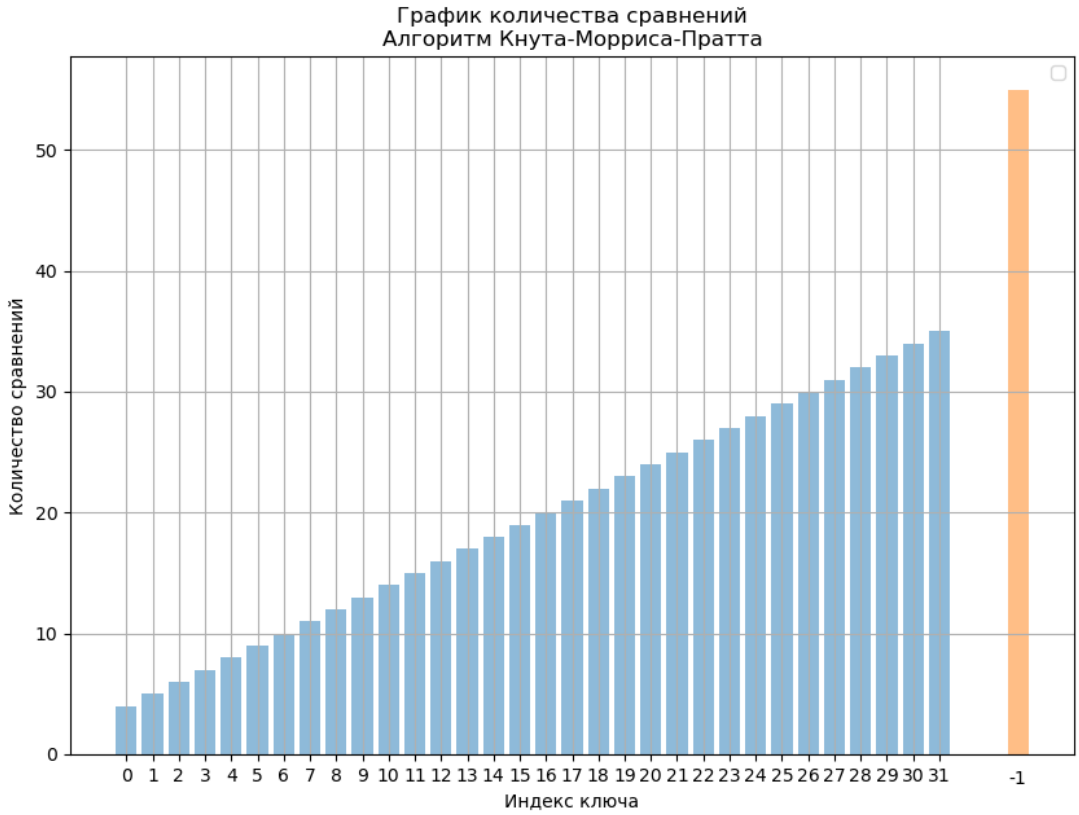

Рисунок 4.3 – Количество сравнений для алгоритма Кнута-Морриса-Пратта

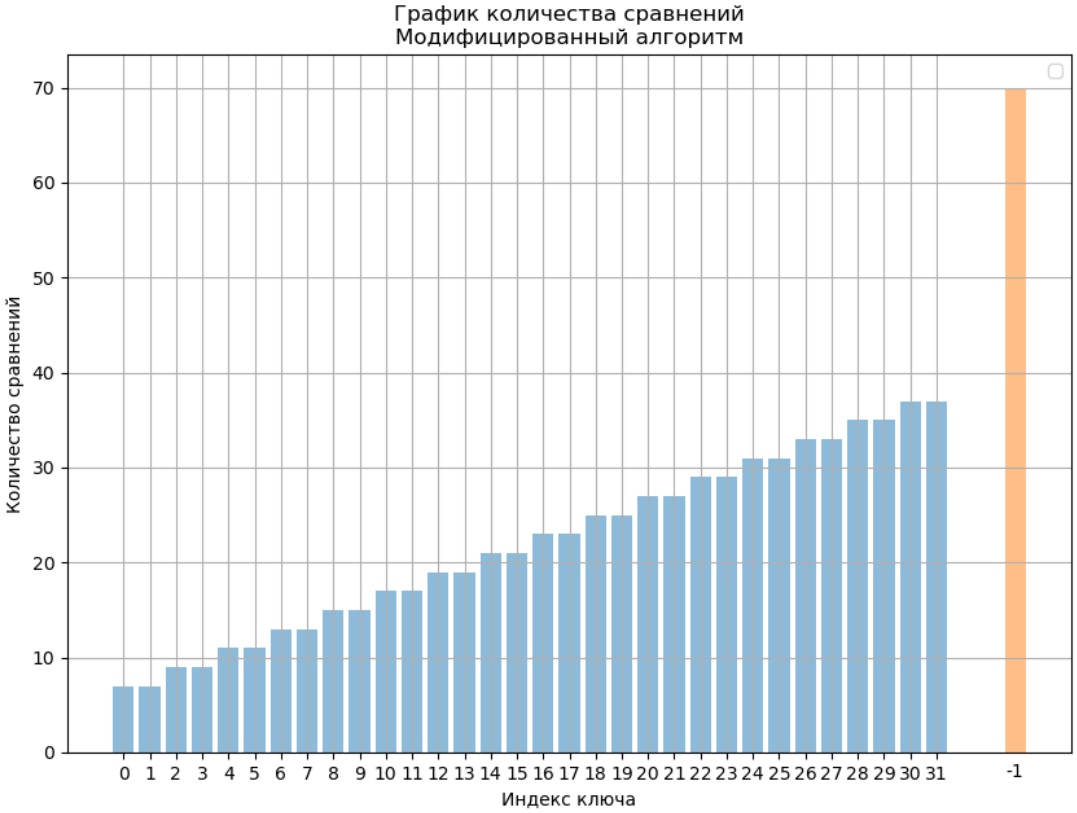

Рисунок 4.4 – Количество сравнений для модифицированного алгоритма

4.5 Вывод

В результате эксперимента было получено, что худший случай для этих алгоритмов — отсутствие подстроки в строке. Реализация алгоритма Кнута-Морриса-Прата работает примерно в 1.5 раза быстрее реализации модифицированного алгоритма, но при этом использует примерно в 2 раза больше операций сравнения.

Заключение

В результате эксперимента было получено, что худший случай для этих алгоритмов — отсутствие подстроки в строке. Реализация алгоритма Кнута-Морриса-Прата работает примерно в 1.5 раза быстрее реализации модифицированного алгоритма, но при этом использует примерно в 2 раза больше операций сравнения.

Цель, которая была поставлена в начале лабораторной работы была достигнута, а также в ходе выполнения лабораторной работы были решены следующие задачи:

— описаны алгоритмы решения задачи поиска подстроки в строке – Кнута-Морриса-Пратта и его модификацию с использованием эвристики «плохого» символа;

— реализованы эти алгоритмы;

— проведен сравнительный анализ рассматриваемых алгоритмов по времени и по количеству сравнений;

— подготовлен отчет по лабораторной работе.

Список литературы

[1] А.Шень. Программирование: теоремы и задачи. М.: МЦНМО. 2021.

[2] Welcome to Python [Электронный ресурс]. Режим доступа: `https://www.python.org` (дата обращения: 23.11.2021).

[3] time — Time access and conversions [Электронный ресурс]. Режим доступа: `https://docs.python.org/3/library/time.html#functions` (дата обращения: 27.11.2021).

[4] Ubuntu 20.04.3 LTS (Focal Fossa) [Электронный ресурс]. Режим доступа: `https://releases.ubuntu.com/20.04/` (дата обращения: 27.11.2021).

[5] Linux [Электронный ресурс]. Режим доступа: `https://www.linux.org/forums/#linux-tutorials.122` (дата обращения: 27.11.2021).

[6] Процессор Intel® Core™ i5-7300HQ [Электронный ресурс]. Режим доступа: `https://ark.intel.com/content/www/ru/ru/ark/products/97456/intel-core-i5-7300hq-processor-6m-cache-up-to-3-50-ghz.html` (дата обращения: 25.11.2021).