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 дисциплине "Анализ алгоритмов"

Тема Поиск в словаре

Студент Хамзина Р. Р.

Группа ИУ7-53Б

Оценка (баллы) _____

Преподаватель Волкова Л. Л.

Москва — 2021 г.

Содержание

Введение

Одной из ключевых операций при обработке данных является поиск. Время работы алгоритмов поиска зависит от размера набора данных. Для представления данных используются различные структуры, которые удобно использовать в конкретных задачах. Одной из таких структур данных является словарь [1]. Следовательно, становится актуальной задача быстрого поиска в словаре.

Целью данной лабораторной работы является изучение алгоритмов поиска в словаре. Для достижения поставленной цели требуется выполнить следующие задачи:

- изучить структуру данных - словарь;
- рассмотреть алгоритмы поиска в словаре: полный перебор, бинарный поиск и поиск сегментами;
- привести схемы изучаемых алгоритмов;
- описать используемые типы и структуры данных;
- описать структуру разрабатываемого программного обеспечения;
- определить средства программной реализации выбранных алгоритмов;
- реализовать разработанные алгоритмы;
- провести функциональное тестирование программного обеспечения;
- провести сравнительный анализ по времени реализованных алгоритмов и числу сравнений при поиске;
- подготовить отчет о выполненной лабораторной работе.

1 Аналитическая часть

В данном разделе будет изучена структура данных словарь и будут рассмотрены алгоритмы поиска в словаре.

1.1 Словарь

Обычные списки (массивы) представляют собой набор пронумерованных элементов, то есть, для обращения к какому-либо элементу списка необходимо указать его номер. Номер элемента в списке однозначно идентифицирует сам элемент. Но идентифицировать данные по числовым номерам не всегда оказывается удобно. Например, маршруты поездов в России идентифицируются численно-буквенным кодом (число и одна цифра), также численно-буквенным кодом идентифицируются авиарейсы, то есть для хранения информации о рейсах поездов или самолетов в качестве идентификатора удобно было бы использовать не число, а текстовую строку.

Структура данных, позволяющая идентифицировать ее элементы не по числовому индексу, а по произвольному, называется словарем или ассоциативным массивом [1]. Данная структура хранит пары вида "ключ-значение". При поиске возвращается значение, которое ассоциируется с данным ключом, или "Пара не найдена", если по данному ключу нет значений.

В данной лабораторной работе используется словарь фильмов и сериалов, в котором:

- ключ - название фильма или сериала;
- значение - его рейтинг.

1.2 Полный перебор

Полный перебор [2] - метод решения, при котором поочередно перебираются все ключи словаря, пока не будет найден нужный.

Чем дальше искомый ключ от начала словаря, тем выше трудоемкость алгоритма. Так, если в начале алгоритм затрагивает b операций, а при сравнении k операций, то:

- элемент найден на первом сравнении за $b + k$ операций (лучший случай);
- элемент найден на i-ом сравнении за $b + i \cdot k$ операций;
- элемент найден на последнем сравнении за $b + N \cdot k$ операций, где N – размер словаря (худший случай);

При этом средняя трудоемкость равна:

$$f = b + k \cdot \left(1 + \frac{N}{2} - \frac{1}{N+1}\right) \tag{1.1}$$

1.3 Бинарный поиск

Бинарный поиск [3] - поиск в заранее отсортированном словаре, который заключается в сравнении элементов со средним на текущем этапе, и, если ключ меньше, то промежутком для поиска становится левая часть, иначе - правая часть.

Тогда случаи расположены следующим образом (b – кол-во операций алгоритма в начале):

- элемент найден на первом сравнении с средним элементом - трудоемкость $b + \log_2 1$ (лучший случай);
- элемент найден на i-ом сравнении - трудоемкость $b + \log_2 i$;
- элемент найден на последнем сравнении - трудоёмкость $b + \log_2 N$, где N - размер словаря (худший случай);

1.4 Сегментный поиск

Идея поиска при помощи сегментов [4] заключается в разбиении словаря на части, в каждую из которых попадают все элементы с некоторым общим

признаком - одинаковый первый символ, цифра, слово.

Обращение к сегменту равно сумме вероятностей обращения к его ключам. Пусть P_i - вероятность обращения к i-ому сегменту, а p_j - вероятность обращения к j-ому элементу i-ого сегмента. Тогда вероятность выбрать нужный сегмент высчитывается так

$$P_i = \sum_j p_j \tag{1.2}$$

Затем ключи в каждом сегменте сортируются, чтобы внутри каждого сегмента можно было произвести бинарный поиск с сложностью $O(\log_2 k)$, где k - количество ключей в сегменте.

То есть, сначала выбирается нужный сегмент, а затем в нем с помощью бинарного поиска ищется нужный ключ.

При этом случаи располагаются так:

- первым выбран верный сегмент, а нужный элемент - серединный (лучший случай);
- нужный сегмент выбран последним, а поиск ключа в данном сегменте - $\log_2 N$, где N - число элементов в сегменте (худший случай);

При этом средняя трудоемкость поиска i-го элемента:

$$\sum_{i \in \Omega} (f_{\text{выбор сегмента i-ого элемента}} + f_{\text{бинарный поиск i-ого элемента}}) \cdot p_i \tag{1.3}$$

1.5 Вывод

Была изучена структура данных - словарь. Были рассмотрены подходы к поиску в словаре.

Программе, реализующей данные алгоритмы, на вход должен подаваться словарь значений и ключ для поиска. Выходными данными такой программы должны быть значение по ключу и число сравнений, выполненных при поиске этого значения. Программа должна работать в рамках следующих ограничений:

- ключ должен быть непустой строкой;
- при пустом словаре или отсутствии данного ключа в словаре должно быть выдано соответствующее сообщение;
- должно быть выдано сообщение об ошибке при некорректном вводе параметров.

Пользователь должен иметь возможность выбора метода решения - полным перебором, бинарным поиском или сегментным поиском, и вывода результата на экран. Также должны быть реализованы сравнение алгоритмов по времени работы и числу сравнений при поиске и получение графического представления результатов сравнения. Данные действия пользователь должен выполнять при помощи меню.

2 Конструкторская часть

В данном разделе будут представлены схемы алгоритмов поиска в словаре: полного перебора, бинарного поиска и поиска сегментами. Будут описаны типы и структуры данных, используемые для реализации, а также структура разрабатываемого программного обеспечения. Кроме того, будут выделены классы эквивалентности для тестирования.

2.1 Разработка алгоритмов

На рисунке 2.1 приведена схема алгоритма поиска в словаре полным перебором. Схема бинарного поиска приведена на рисунке 2.2. Схема сегментного поиска показана на рисунке 2.3.

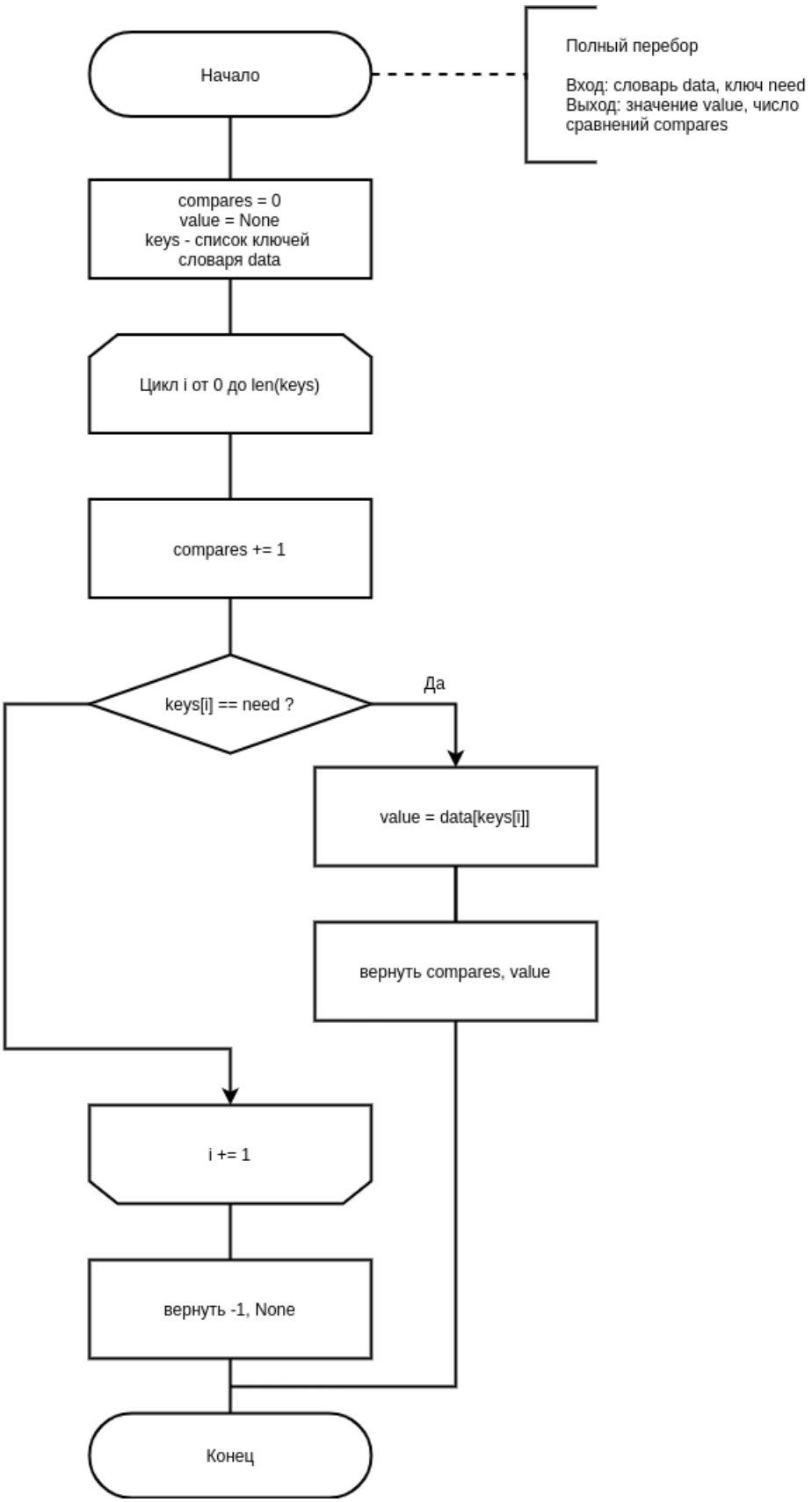

Рисунок 2.1 – Полный перебор

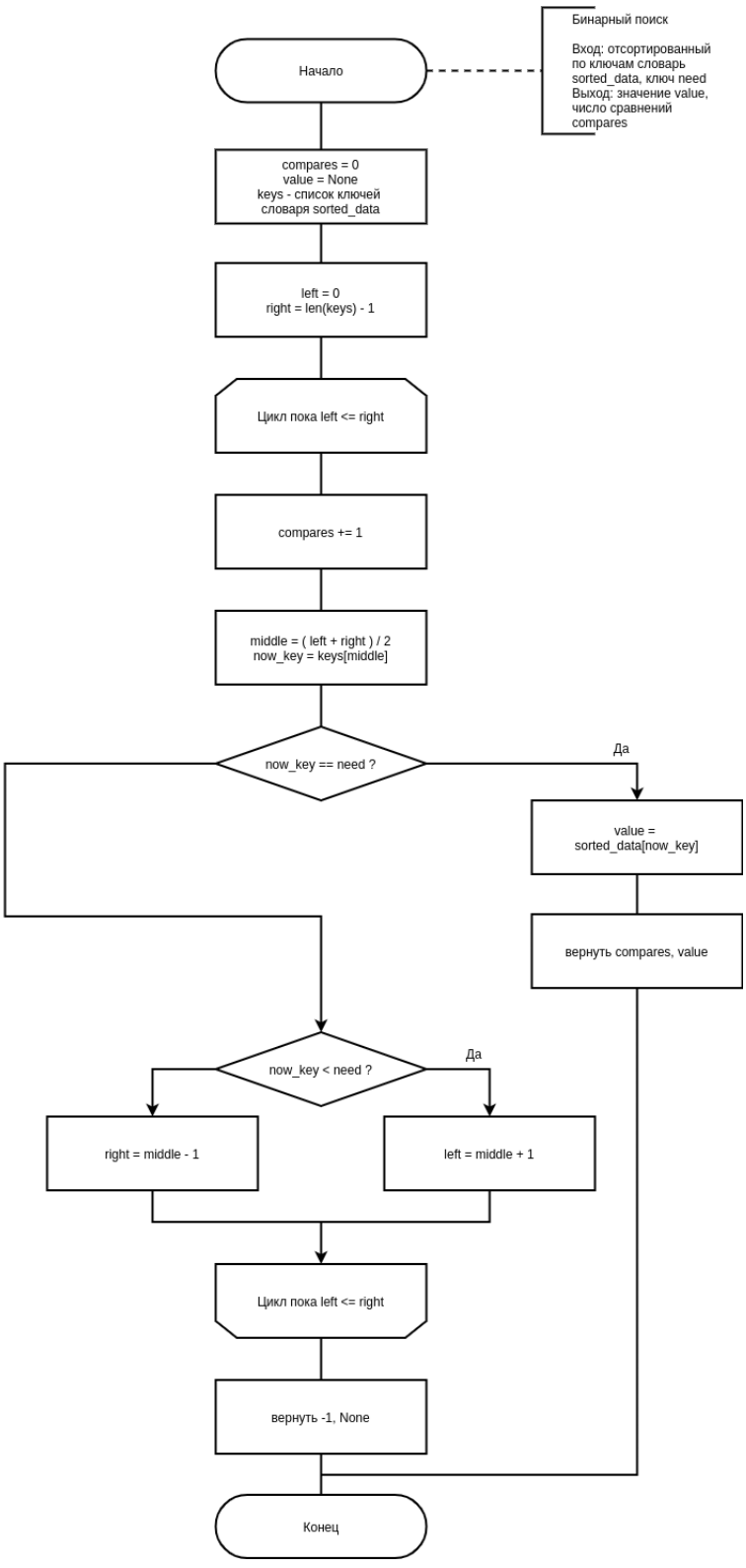

Рисунок 2.2 – Бинарный поиск

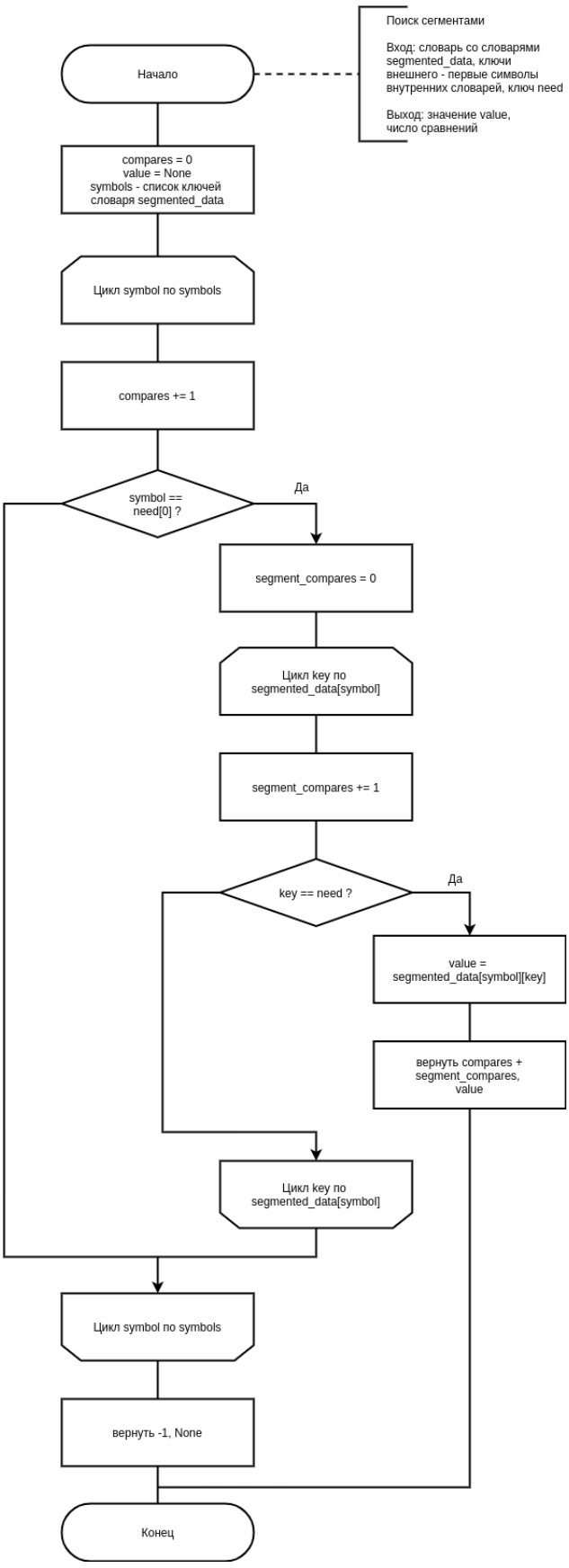

Рисунок 2.3 – Поиск сегментами

2.2 Описание используемых типов и структур данных

Для реализации словаря будет использован встроенный тип данных *dict* в *Python*, при этом для задания ключа и значения будет использован тип данных *str*.

Для числа сравнений будет использован тип данных *int*.

2.3 Структура разрабатываемого ПО

При реализации разрабатываемого программного обеспечения будет использоваться метод структурного программирования. Для взаимодействия с пользователем будет выделена функция *process*(), из которой будут вызываться методы поиска в словаре и функции сравнительного анализа. Методы поиска в словаре будут реализованы в следующих функциях, входные параметры которых - ключ, по которому происходит поиск, и словарь значений, а выходные параметры - число сравнений и найденное по ключу значение:

- функция полного перебора;
- функция бинарного поиска;
- функция поиска сегментами.

Для работы со словарем будут разработаны следующие функции:

- загрузка данных в словарь из файла, входной параметр - имя файла, выходный - заполненный словарь значений;
- сортировка словаря по ключам, входной параметр - словарь значений, выходной - отсортированный по ключам словарь;
- сортировка словаря по значениям, входной параметр - словарь значений, выходной - отсортированный по значениям словарь;

- вывод словаря, входной параметр - словарь значений, выходной - вывод пар ключ-значение на экран.

Для сравнительного анализа будут реализованы:

- функция замеров времени, выходным параметром которой является массив временных значений;
- функция замеров числа сравнений, выходным параметром которой является массив числа сравнений;
- функция графического представления замеров времени, у которой на входе - массив временных значений или массив числа сравнений, на выходе - графическое представление.

2.4 Классы эквивалентности при тестировании

Для тестирования разрабатываемой программы будут выделены следующие классы эквивалентности:

- неверный ввод ключа - пустое значение;
- введенного ключа нет в словаре;
- введенный ключ есть в словаре.

2.5 Вывод

Были представлены схемы поиска в словаре. Были указаны типы и структуры данных, используемые для реализации, и описана структура разрабатываемого программного обеспечения. Также были выделены классы эквивалентности для тестирования ПО.

3 Технологическая часть

В данном разделе будут указаны средства реализации, будут представлены листинги кода, а также функциональные тесты.

3.1 Средства реализации

Реализация данной лабораторной работы выполнялась при помощи языка программирования Python [5]. Выбор ЯП обусловлен простотой синтаксиса, большим числом библиотек и эффективностью визуализации данных.

Замеры времени проводились при помощи функции process_time из библиотеки time [6].

3.2 Сведения о модулях программы

Программа состоит из следующих модулей:

- main.py - главный файл программы, предоставляющий пользователю меню для выполнения основных функций;
- dictionary.py - файл, содержащий функции работы со словарями и методы поиска в словаре;
- test_func.py - файл, содержащий функции сравнительного анализа указанных алгоритмов;
- graph_result.py - файл, содержащий функции визуализации сравнительного анализа описанных алгоритмов.

3.3 Листинги кода

Поиск полным перебором приведен в листинге 3.1, бинарный поиск - в листнге 3.2, сегментный поиск - в листинге 3.3.

Листинг 3.1 – Полный перебор

```
def brute_force(self, need):

    compares = 0
    value = None

    keys = self.data.keys()

    for key in keys:
        compares += 1

        if key == need:
            value = self.data[key]
            return compares, value

    return Dictionary.NO_RESULT, value
```

Листинг 3.2 – Бинарный поиск

```
def binary_search(self, need, sorted_data):
    compares = 0
    value = None

    keys = list(sorted_data)
    left = 0
    right = len(keys) - 1

    while left <= right:
        compares += 1
        middle = (left + right) // 2
        now_key = keys[middle]

        if now_key < need:
            left = middle + 1
        elif now_key > need:
            right = middle - 1
        else:
            value = sorted_data[now_key]
            return compares, value

    return Dictionary.NO_RESULT, value
```

Листинг 3.3 – Сегментный поиск

```
def segment_data(self):
    segments = {symbol: 0 for symbol in "ABCDEFGHIJMS12345679"}

    for key in self.data:
        segments[key[Dictionary.FIRST_SYMBOL]] += 1

    segments = self.sort_values(segments)

    segmented_data = {pair: dict() for pair in segments}

    for key in self.data:
        segmented_data[key[Dictionary.TITLE]].update({key:
            self.data[key]})

    return segmented_data

def segment_search(self, need, segmented_data):
    compares = 0
    value = None

    symbols = list(segmented_data.keys())

    for symbol in symbols:
        compares += 1

        if symbol == need[Dictionary.FIRST_SYMBOL]:
            segment_compares = 0

            for key in segmented_data[symbol]:
                segment_compares += 1

                if key == need:
                    value = segmented_data[symbol][key]
                    return compares + segment_compares, value

            return Dictionary.NO_RESULT, value


    return Dictionary.NO_RESULT, value
```

3.4 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для функций программы. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Ключ	Тип поиска	Ожидаемый результат
Пустой ввод	Любой	Сообщение об ошибке
A	Любой	Фильм не найден.
37 Seconds	Полный перебор	37 Seconds —> TV-MA, 90
Friends	Бинарный	Friends —> TV-14, 11
1983	Сегментный	1983 —> TV-MA, 19

3.5 Вывод

Были реализованы функции алгоритмов поиска в словаре. Было проведено функциональное тестирование указанных функций.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программы, и будет проведен сравнительный анализ реализованных алгоритмов поиска в словаре по затраченному процессорному времени и по числу сравнений при поиске.

4.1 Технические характеристики

Тестирование проводилось на устройстве со следующими техническими характеристиками:

- операционная система: Ubuntu 20.04.1 Linux x86_64 [7];
- память : 8 GiB;
- процессор: AMD® Ryzen™ 3 3200u @ 2.6 GHz [8].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.2 Демонстрация работы программы

На рисунке 4.1 приведен пример работы программы при наличии введенного ключа в словаре, на рисунке 4.2 - при отсутствии введенного ключа в словаре.

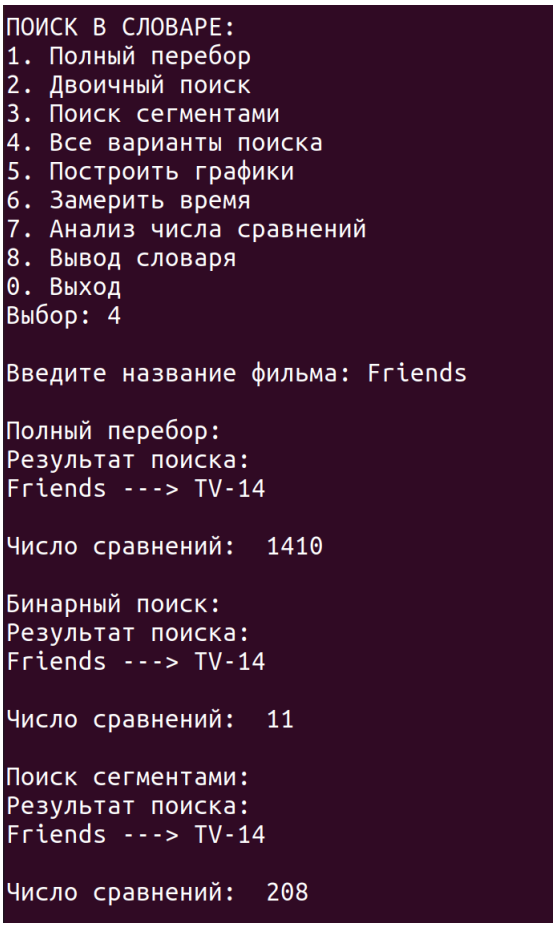

Рисунок 4.1 – Пример работы программы (ключ найден)

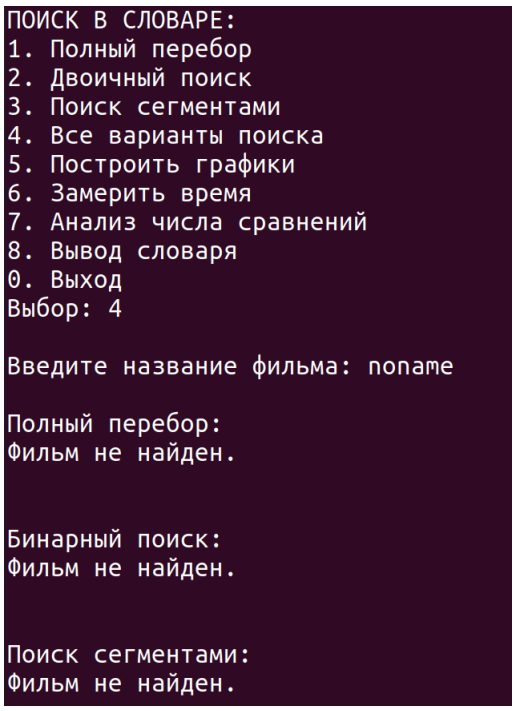

Рисунок 4.2 – Пример работы программы (ключ не найден)

4.3 Время выполнения алгоритмов

Функция process_time из библиотеки time ЯП Python возвращает процессорное время в секундах - значение типа float.

Для замера времени необходимо получить значение времени до начала выполнения алгоритма, затем после её окончания. Чтобы получить результат, необходимо вычесть из второго значения первое. Замеры проводились 40 раз для получения точного результата.

Замеры проводились для всех ключей словаря. Графическое представление результатов измеренения времени работы алгоритмов приведено на рисунке 4.3.

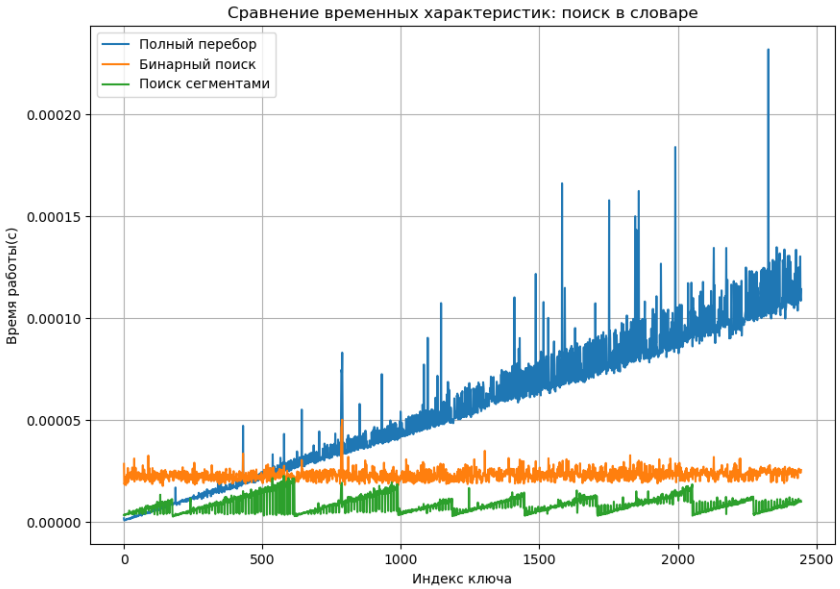

Рисунок 4.3 – Сравнительный анализ алгоритмов поиска в словаре по времени

4.4 Число сравнений при поиске в словаре

Для всех алгоритмов поиска в словаре был проведен сравнительный анализ по числу сравнений для всех ключей в словаре. Были сопоставлены две гистограммы для каждого из алгоритмов поиска:

- ключи расположены в порядке их расположения в словаре;
- ключи отсортированы по убыванию числа сравнений.

Полученные гистограммы для поиска полным перебором представлены на рисунке 4.4, для бинарного поиска - на рисунке 4.5, для поиска сегментами - на рисунке 4.6.

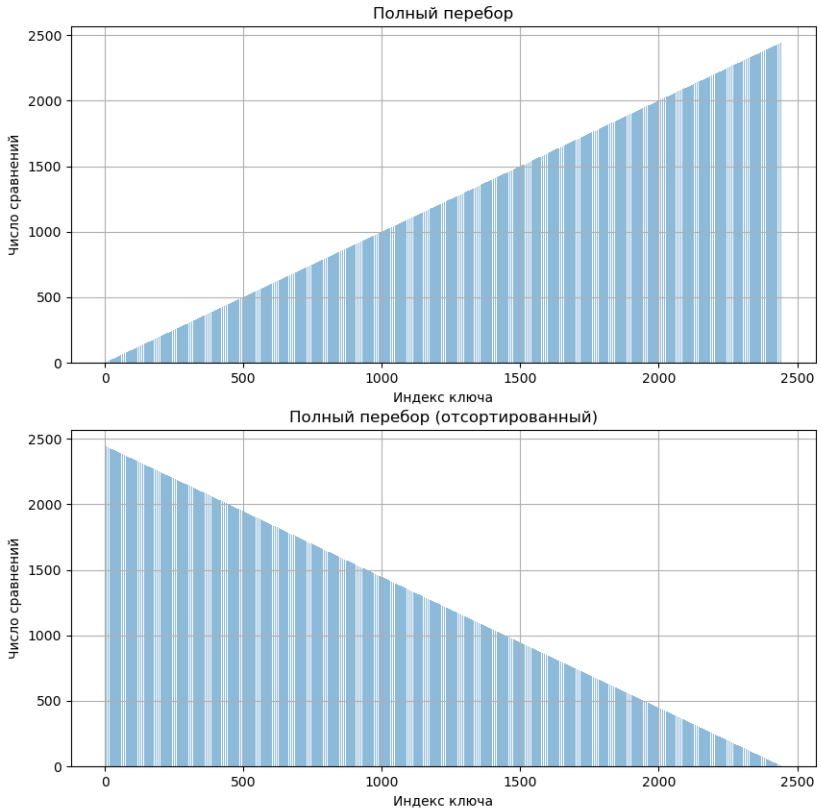

Рисунок 4.4 – Сравнительный анализ алгоритмов поиска в словаре по числу сравнений - полный перебор

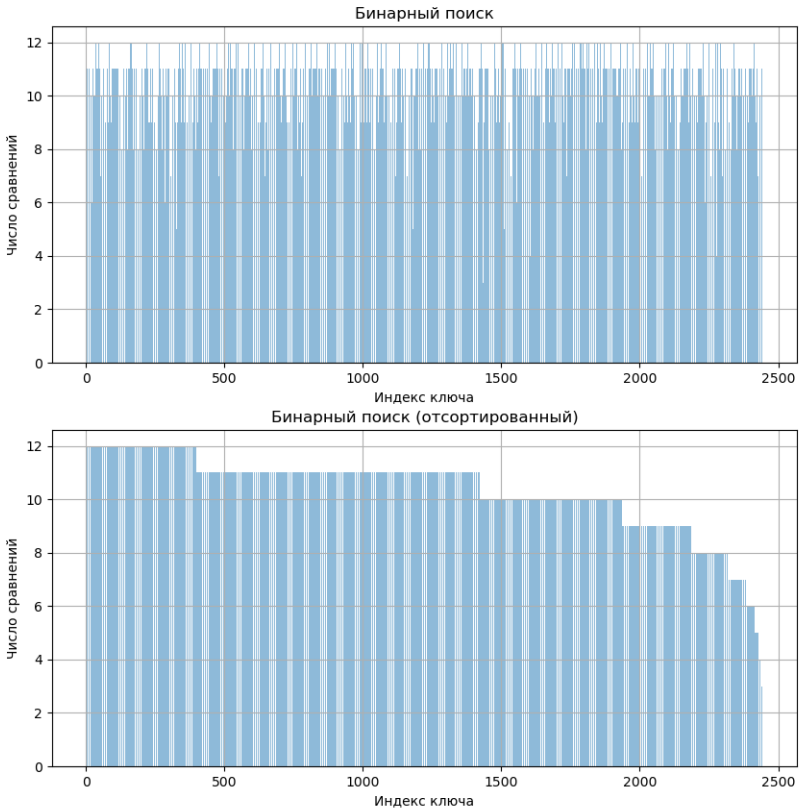

Рисунок 4.5 – Сравнительный анализ алгоритмов поиска в словаре по числу сравнений - бинарный поиск

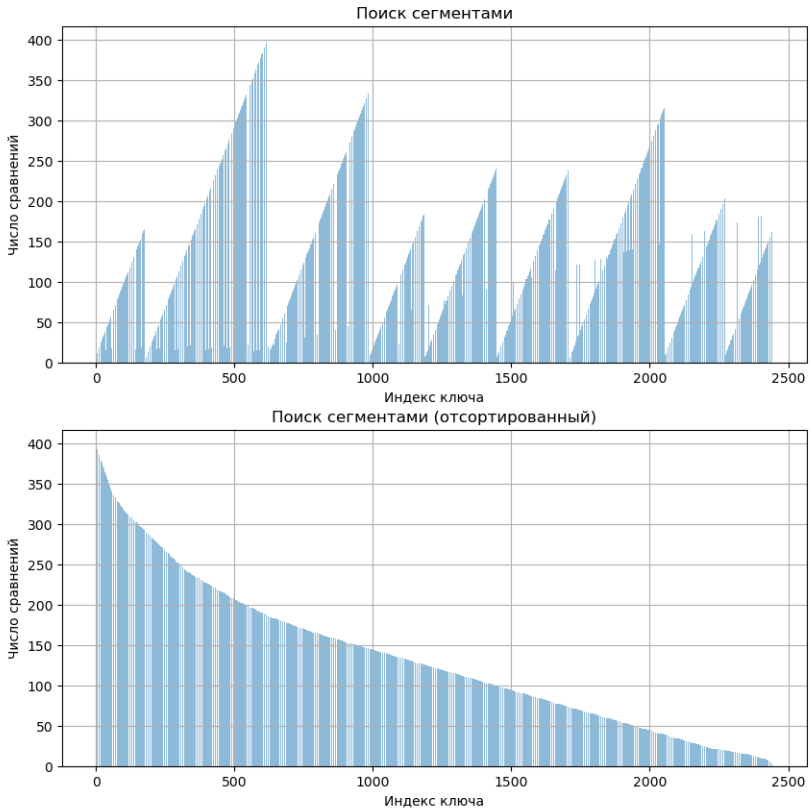

Рисунок 4.6 – Сравнительный анализ алгоритмов поиска в словаре по числу сравнений - сегментный поиск

4.5 Вывод

В результате эксперимента было получено:

- время работы поиска при помощи полного перебора увеличивется пропорционально дальности ключа в словаре;
- бинарный и сегментный поиск имеют в среднем одинаковое время поиска всех ключей;
- при большом количестве ключей, близком к 2500, сегментный поиск работает быстрее полного перебора в 14 раз, бинарный поиск работает быстрее полного перебора в 6 раз.

Можно сделать вывод о том, что при обработке больших данных (близком к 2500) необходимо использовать поиск сегментами.

Также в результате сравнительного анализа было получено, что алгоритму бинарного поиска требуется меньшее число сравнений в сравнении с полным перебором и сегментным поиском. Для поиска в словаре значений, используемом при выполнении данной работы, максимальные числа сравнений:

- для бинарного поиска - 12;
- для сегментного поиска - 398;
- для полного перебора - 2446.

Таким образом, в задачах с ограничением по числу сравнений при поиске необходимо использовать бинарный поиск в словаре.

Заключение

В результате исследования было получено, что при большом числе ключей в словаре (близком к 2500) следует использовать сегментный поиск, так как он быстрее в 14 раз поиска полным перебором, время работы которого зависит от дальности расположения ключа в словаре. При этом в задачах с ограничением на число сравнений при поиске необходимо использовать бинарный поиск, которому для данного словаря потребовалось максимум 12 сравнений, в то время как для сегментного поиска максимум потребовалось 398 сравнений, для полного перебора - 2446.

Цель, поставленная перед началом работы, была достигнута. В ходе лабораторной работы были решены следующие задачи:

- был рассмотрен поиск в словаре и алгоритмы его реализации;
- были разработаны линейный поиск, бинарный поиск и поиск сегментами;
- был проведен сравнительный анализ реализованных алгоритмов;
- был подготовлен отчет о выполненной лабораторной работе.

Список литературы

[1] Словари [Электронный ресурс]. Режим доступа: https://informatics.msk.ru/mod/book/view.php?id=6694 (дата обращения: 10.12.2021).

[2] Полный перебор [Электронный ресурс]. Режим доступа: http://skud-perm.ru/posts/polnyj-perebor (дата обращения: 10.12.2021).

[3] Бинарный поиск [Электронный ресурс]. Режим доступа: https://prog-cpp.ru/search-binary/ (дата обращения: 10.12.2021).

[4] Нильсон Н. Искусственный интеллект. Методы поиска решений. 1973.

[5] Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org (дата обращения: 10.12.2021).

[6] time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html#functions (дата обращения: 10.12.2021).

[7] Ubuntu 20.04 LTS (Focal Fossa) Beta [Электронный ресурс]. Режим доступа: http://old-releases.ubuntu.com/releases/20.04.1/ (дата обращения: 10.12.2021).

[8] Мобильный процессор AMD Ryzen™ 3 3200U с графикой Radeon™ Vega 3 [Электронный ресурс]. Режим доступа: https://www.amd.com/ru/products/apu/amd-ryzen-3-3200u (дата обращения: 10.12.2021).