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 Поиск в словаре

Студент Цветков И.А.

Группа ИУ7-53Б

Оценка (баллы)

Преподаватель Волкова Л. Л.

Москва — 2021 г.

Содержание

Введение

В процессе развития компьютерных систем количество данных стало достигать огромных размеров, поэтому множество операций стали выполняться очень долго, поскольку чаще всего это был обычный перебор. Это вызвало необходимость создать новые алгоритмы, которые решают поставленную задачу на порядок быстрее стандартного решения “в лоб”. В том числе это касается и словарей, в которых одной из основных операций является операция поиска.

Целью данной работы является изучение алгоритмов поиска в словаре – полным перебором, бинарным поиском и сегментами. Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить понятие словаря;
- описать алгоритмы решения задачи поиска в словаре – полный перебор, бинарный поиск и сегментами;
- привести схемы алгоритмов;
- описать используемые типы и структуры данных;
- описать структуру разрабатываемого программного обеспечения;
- реализовать разработанные алгоритмы;
- провести функциональное тестирование разработанного алгоритма;
- провести сравнительный анализ по времени для реализованного алгоритма;
- подготовить отчет по лабораторной работе.

1 Аналитическая часть

В этом разделе будет представлена информация о словаре, а также об алгоритмах поиска в нем – полным перебором, бинарном и сегментами.

1.1 Словарь

Словарь [1] – тип данных, который позволяет хранить пары вида "ключ-значение" – *(k, v)*. Он поддерживает три операции – добавление пары, поиск по ключу, удаление по ключу. В паре *(k, v)* – v это значение, которое ассоциируется с ключом v.

При поиске возвращается значение, которое ассоциируется с данным ключом, или "не найдено", если по данному ключу нет значений.

В данной лабораторной работе:

- ключ – фамилия футболиста;
- значение – информация о нем.

1.2 Алгоритм полного перебора

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 алгоритм. Так, если на старте 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 со средним элементов, и, если ключ меньше, то продолжать поиск в левой части тем же методом, иначе – в правой части.

Тогда случаи расположены следующим образом (b – кол-во операций алгоритма на старте):

- 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 Поиск с помощью сегментов

Поиск с помощью сегментов [4] – словарь разбивается на части, в каждую из которых попадают все элементы с некоторым общим признаком – одинаковая первая буква, 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

В данном разделе было рассмотрено понятие словаря, а также алгоритма поиска в словаре – полным перебором, бинарный и сегментами.

Программа будет получать на вход словарь, а также ключ, по которому нужно будет найти значение в данном словаре одним из трех алгоритмов, который также будет вводиться. Если словарь пуст или какое-то из значений введено неверно, то будет выдано сообщение об ошибке.

Реализуемое ПО дает возможность получить значение по ключу одним из трех алгоритмов поиска в словаре. Также имеется возможность провести тестирование по времени для рассматриваемых алгоритмов.

2 Конструкторская часть

В этом разделе будут представлено описание используемых типов данных, а также схемы алгоритма полного перебора, бинарного поиска и поиска разбиением на сегменты.

2.1 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие типы данных:

- название файла - строка типа *str*;
- список ключей или значений - список типа *str*;
- словарь для хранения информации о футболисте по его имени (ключу) - встроенный тип *dict* в *Python*.

2.2 Структура разрабатываемого ПО

В данном ПО будет реализован метод структурного программирования. Для взаимодействия с пользователем будет разработано меню, которое будет предоставлять возможность выбрать нужный алгоритм - полного перебора, бинарного поиска или поиска сегментами, а также построить графики для сравнения времени выполнения алгоритмов и гистограммы для анализа количества сравнений для каждого алгоритма.

Для работы будут разработаны следующие процедуры:

- процедура, реализующая создание словаря, используя информацию из файла, входные данные - имя файла, выходные - заполненный словарь;
- процедура вывода полученного словаря на экран (для отладки), входные данные - словарь, выходные - выведенный на экран словарь;

- процедура, реализующая алгоритм поиска полным перебором, входные данные - словарь значений, ключ (по которому будет производиться поиск информации), выходные - количество сравнений, а также найденное по ключу значение в словаре;

- процедура, реализующая алгоритм бинарного поиска, входные данные - словарь значений, ключ (по которому будет производиться поиск информации), выходные - количество сравнений, а также найденное по ключу значение в словаре;

- процедура, реализующая алгоритм поиска сегментами, входные данные - словарь значений, ключ (по которому будет производиться поиск информации), выходные - количество сравнений, а также найденное по ключу значение в словаре;

- процедуры замера времени алгоритмов полного перебора, бинарного поиска и поиска сегментам для всех ключей словаря, входные данные - словарь, выходные - результаты замеров времени;

- процедура для построения гистограмм для полученных значений сравнений для каждого ключа в словаре, входные данные - словарь, выходные - гистограммы (отсортированная (по убыванию количества сравнений) и нет);

- процедура для построения графиков по полученным временным замерам, входные данные - замеры времени, выходные - график.

2.3 Схемы алгоритмов

На рисунке 2.1 представлена схема алгоритма поиска в словаре с помощью полного перебора, а на рисунках 2.2 и 2.3 схемы алгоритмов бинарного и сегментного поисков соответственно.

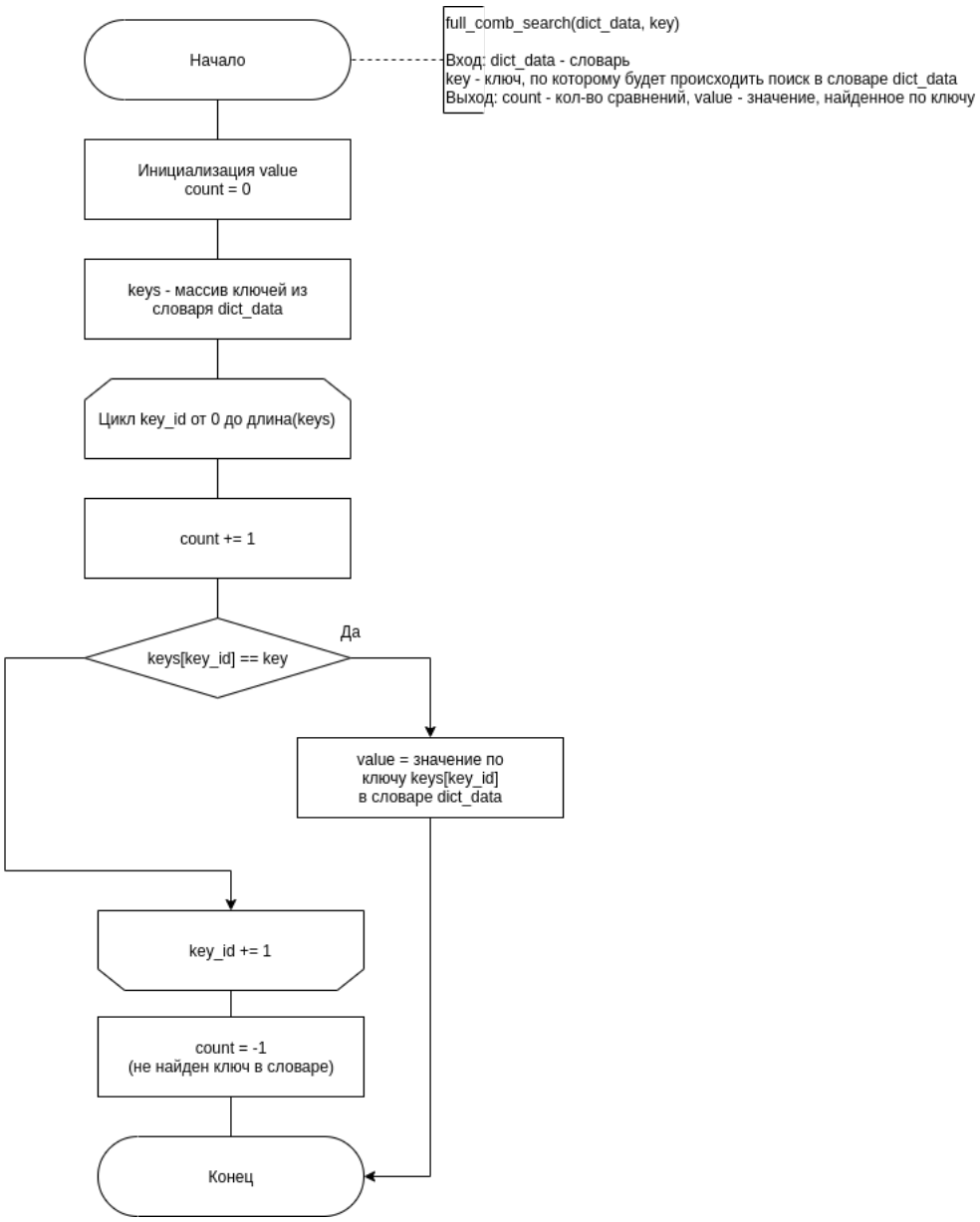

Рисунок 2.1 – Схема алгоритма поиска полным перебором

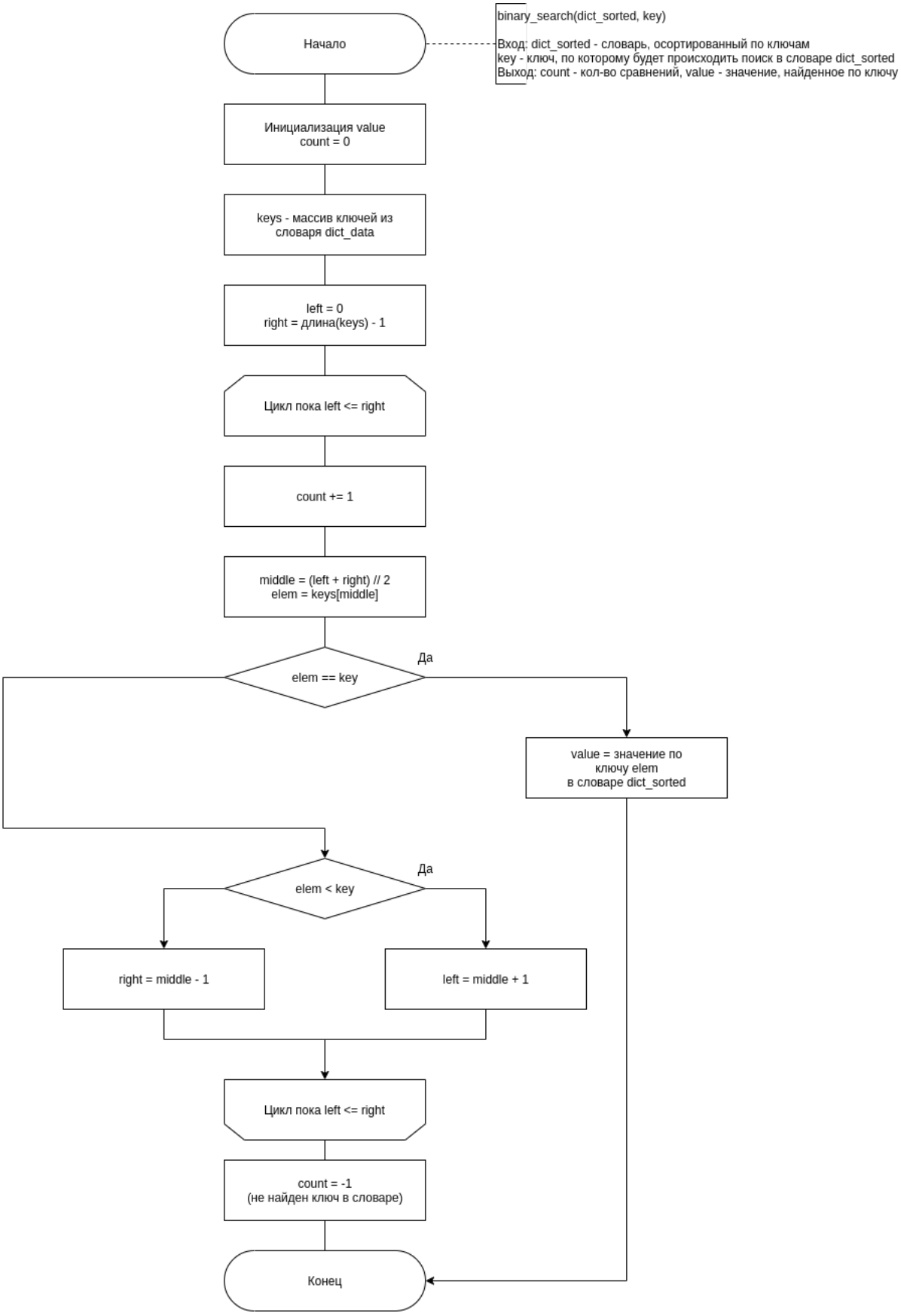

Рисунок 2.2 – Схема алгоритма бинарного поиска

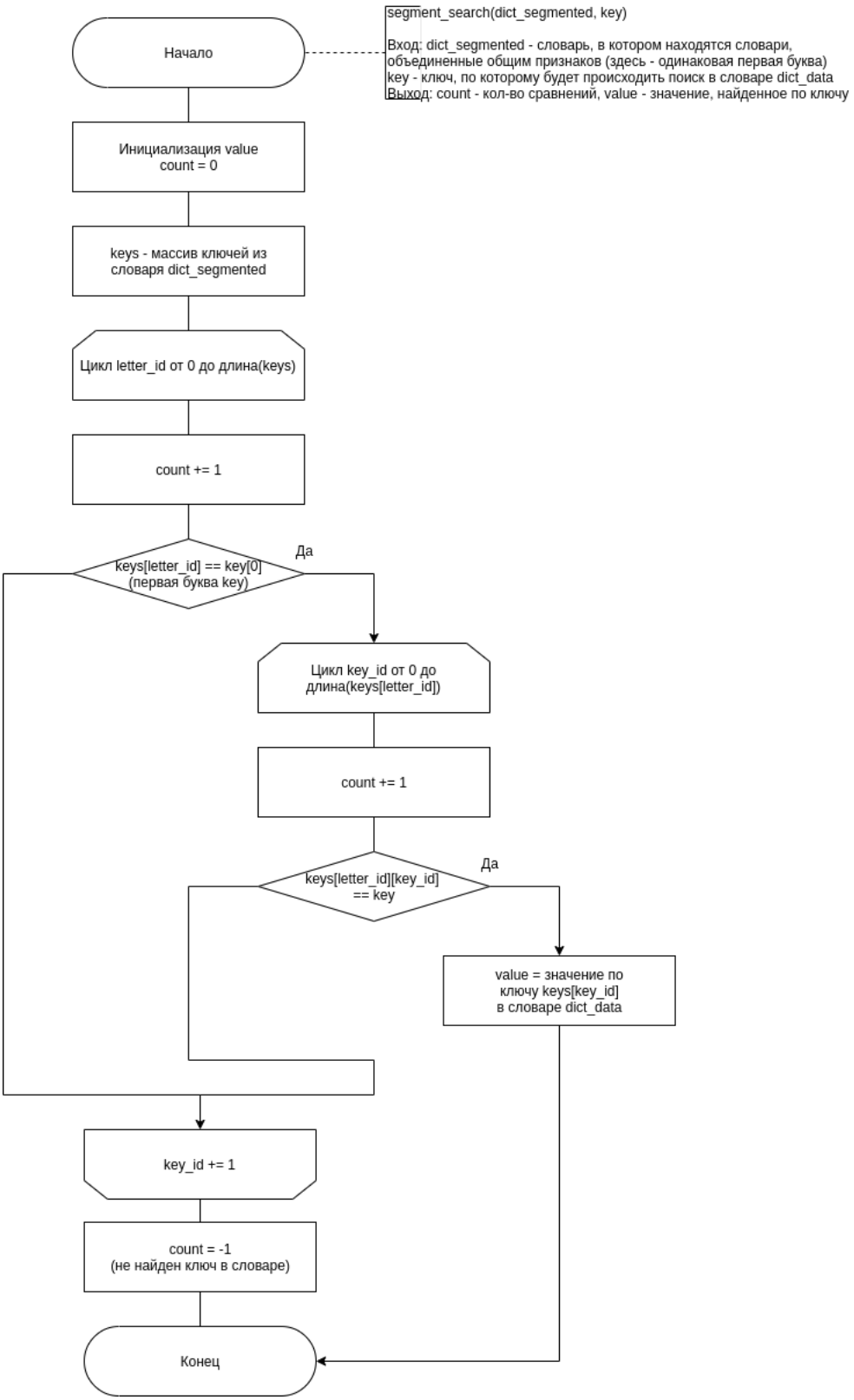

Рисунок 2.3 – Схема алгоритма поиска с разбиением словаря на сегменты

2.4 Классы эквивалентности при тестировании

Для тестирования выделены классы эквивалентности, представленные ниже.

1. Неверно выбран пункт меню - не число или число, меньшее 0 или большее 7.
2. Неверно введен ключ - пустая строка.
3. Введенного ключа нет в словаре.
4. Введенный ключ есть в словаре.

2.5 Вывод

В данном разделе были построены схемы алгоритмов, рассматриваемых в лабораторной работе, были описаны классы эквивалентности для тестирования, структура программы.

3 Технологическая часть

В данном разделе будут рассмотрены средства реализации, а также представлены листинги алгоритма поиска полным перебором, бинарного алгоритма поиска и алгоритма поиска с использованием сегментации.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования *Python*[5]. В текущей лабораторной работе требуется замерить процессорное время для выполняемой программы, а также построить графики. Все эти инструменты присутствуют в выбранном языке программирования.

Время работы было замерено с помощью функции *process_time(...)* из библиотеки *time*[6].

3.2 Листинги кода

В листинге 3.1 представлен алгоритм поиска полного перебора в словаре, а в листинге 3.2 – алгоритм бинарного поиска. Также в листинге 3.3 представлен алгоритм разбиения словаря на сегменты по общему признаку – ключи, начинающиеся на одну букву, а в листинге 3.4 – поиск в словаре сегментами.

Листинг 3.1 – Алгоритм поиска в словаре полным перебором

```
def full_search(data_dict, key, output = True):
    count = 0 # count of comparsions

    keys = data_dict.keys()

    for elem in keys:
        count += 1

        if (elem == key):
            if (output):
                record = [key, data_dict[key]]
                print("\n\nSearch result:\n")

                print_record(record)

            return count

    return -1
```

Листинг 3.2 – Алгоритм бинарного поиска

```
def binary_search(sorted_dict, key, output = True):
    count = 0 # count of comparsions

    keys = list(sorted_dict.keys())

    left = 0
    right = len(keys)

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

        if (elem == key):
            if (output):
                record = [key, sorted_dict[key]]
                print("\n\nSearch result:\n")
                print_record(record)

            return count

        if (elem < key):
            left = middle + 1
        else:
            right = middle - 1

    return -1
```

Листинг 3.3 – Алгоритм разбиения словаря на сегменты

```
def make_segments(self):
    temp_dict = {i: 0 for i in "ABCDEFGHIJKLMNOPQRSTUVWXYZ"}

    for key in self.data_dict:
        temp_dict[key[FIRST_LETTER]] += 1

    temp_dict = self.sort_value(temp_dict)

    segmented_dict = {i: dict() for i in temp_dict}

    for key in self.data_dict:
        segmented_dict[key[0]].update({key: self.data_dict[key]})

    return segmented_dict
```

Листинг 3.4 – Алгоритм поиска в словаре с использованием сегментации

```
def segment_search(segmented_dict, key, output = True):
    count = 0

    keys = list(segmented_dict.keys())

    for key_letter in keys:
        count += 1

        if (key[FIRST_LETTER] == key_letter):
            count_search = 0

            for elem in segmented_dict[key_letter]:
                count_search += 1

                if (elem == key):
                    if (output):
                        record = [key,
                           segmented_dict[key_letter][key]]
                        print("\n\nSearch result:\n")
                        print_record(record)

                    return count_search + count

            return -1

    return -1
```

3.3 Сведения о модулях программы

Программа состоит из двух модулей:

- *main.py* - файл, содержащий меню программы, а также весь служебный код;
- *dictionary.py* - файл, содержащий реализацию класса *Dictionary*, в котором содержатся методы для работы со словарем, а также алгоритмы поиска – полным перебором, бинарный, сегментами.

3.4 Функциональные тесты

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

Таблица 3.1 – Функциональные тесты

Ключ	**Результат**	**Пояснение**
Noname	Нет такого ключа	Несуществующий ключ
Messi	Информация и кол-во сравнений	Первый ключ
Bonifazi	Информация и кол-во сравнений	Крайний ключ
Golovin	Информация и кол-во сравнений	Ключ в словаре

3.5 Вывод

Были представлены листинги всех алгоритмов – полного перебора, бинарного поиска, сегментного поиска. Также в данном разделе была приведена информации о выбранных средствах для разработки алгоритмов и сведения о модулях программы, проведено функциональное тестирование.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программа, а также проведен сравнительный анализ алгоритмов при различных ситуациях на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование представлены далее:

- операционная система: Ubuntu 20.04.3 [7] Linux [8] x86_64;
- память: 8 GiB;
- процессор: Intel® Core™ i5-7300HQ CPU @ 2.50GHz [9].

При тестировании ноутбук был включен в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также системой тестирования.

4.2 Демонстрация работы программы

На рисунке 4.1 представлен результат работы программы для всех алгоритмов – поиск полным перебором, бинарного поиска и поиска сегментами.

```
     1. Полный перебор
     2. Бинарный поиск
     3. Поиск сегментами

     4. Все алгоритмы
     5. Замерить время
     6. Проанализировать кол-во сравнений

     7. Распечатать словарь

     0. Выход

         Выбор:     4

Введите фамилию футболиста: Golovin


Результат поиска:

Name: Golovin
Rating: 78, Country: Russia, Club: AS Monaco

Полный перебор:
Количество сравнений:  522


Результат поиска:

Name: Golovin
Rating: 78, Country: Russia, Club: AS Monaco

Бинарный поиск:
Количество сравнений:  12


Результат поиска:

Name: Golovin
Rating: 78, Country: Russia, Club: AS Monaco

Поиск сегментами:
Количество сравнений:  37
```

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.

Использовать функцию приходится дважды, затем из конечного времени нужно вычесть начальное, чтобы получить результат.

Замеры проводились для всех ключей из словаря по 30 раз, чтобы добиться более точного значения времени работы каждого из алгоритмов.

Результаты замеров приведены на рисунке 4.2 в графическом виде.

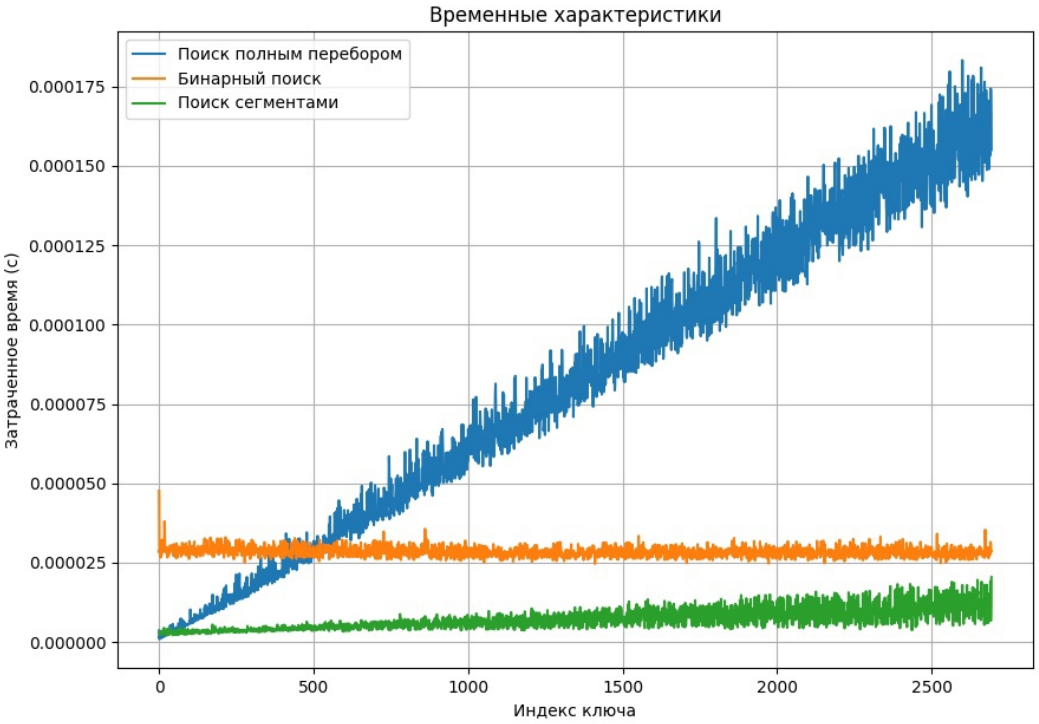

Рисунок 4.2 – Сравнение по времени алгоритмов поиска полным перебором, бинарного поиска и поиска сегментами

4.4 Количество сравнений при работе алгоритмов

Для каждого алгоритма был проведен анализ по количеству сравнений для нахождения каждого ключа в словаре. При этом были составлены по две гистограммы для всех алгоритмов:

- ключи расположены в том же порядке, как и в словаре;
- ключи отсортированы в порядке убывания кол-ва сравнений.

На рисунке 4.3 представлены гистограммы для поиска полным перебором, на рисунке 4.4 – бинарного поиска, а на рисунке 4.5 – поиск сегментами.

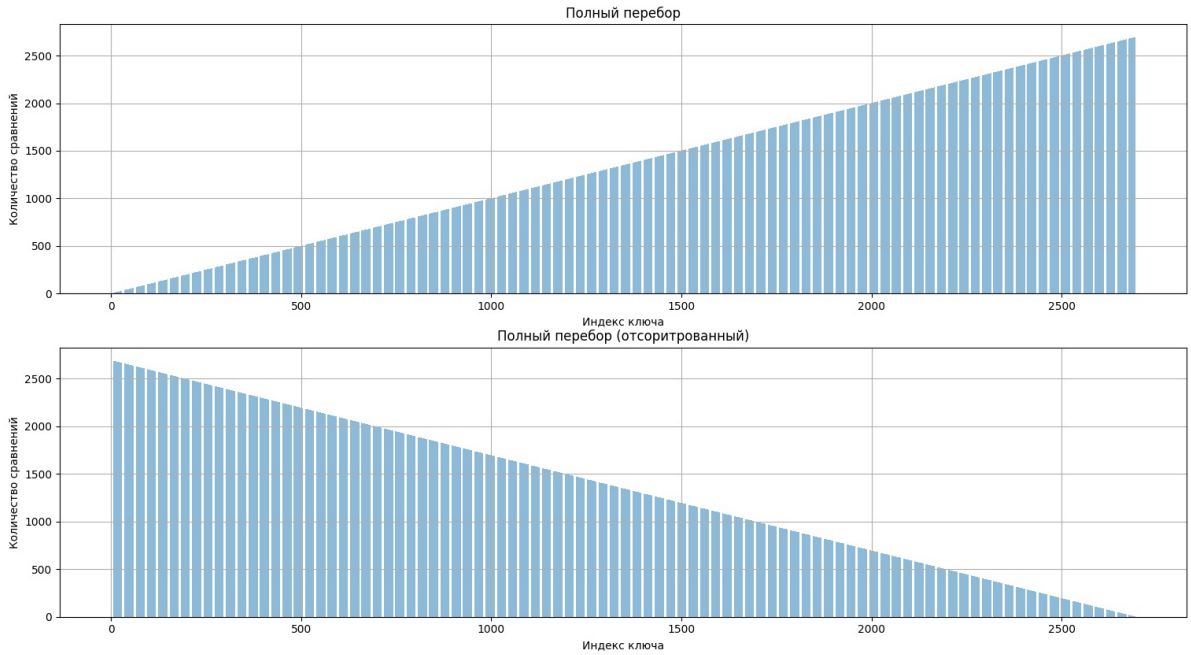

Рисунок 4.3 – Сравнение по количеству сравнений при поиске ключа в словаре алгоритмом полного перебора

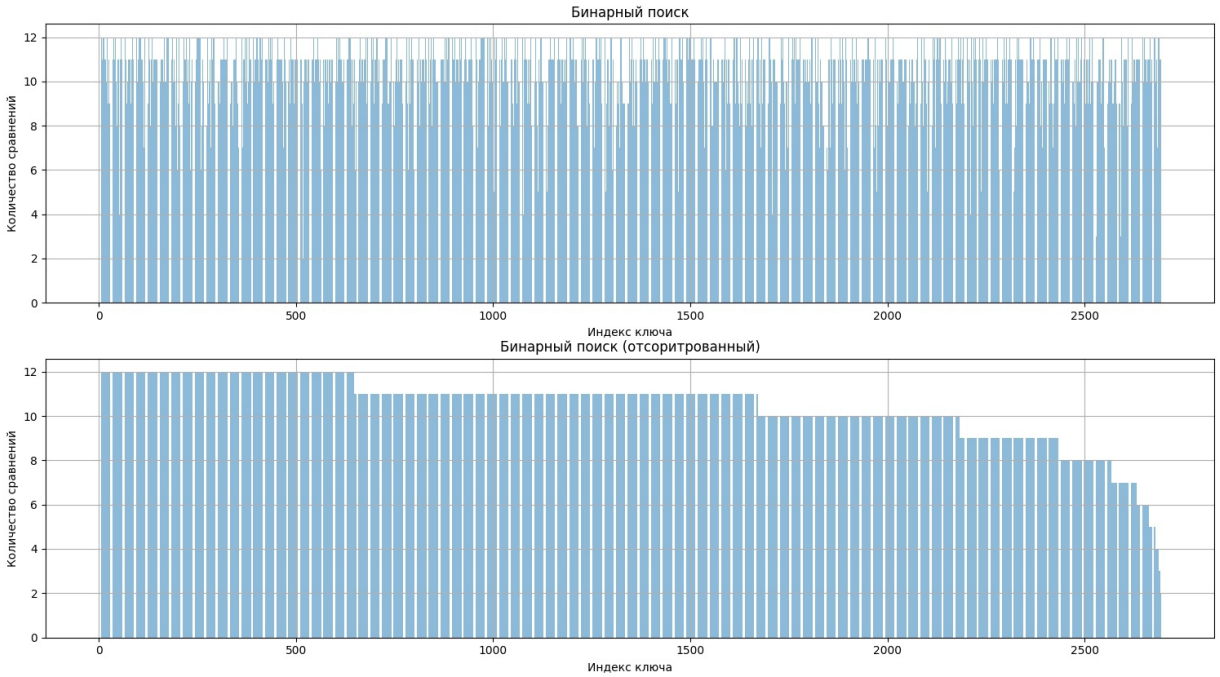

Рисунок 4.4 – Сравнение по количеству сравнений при поиске ключа в словаре бинарным алгоритмом

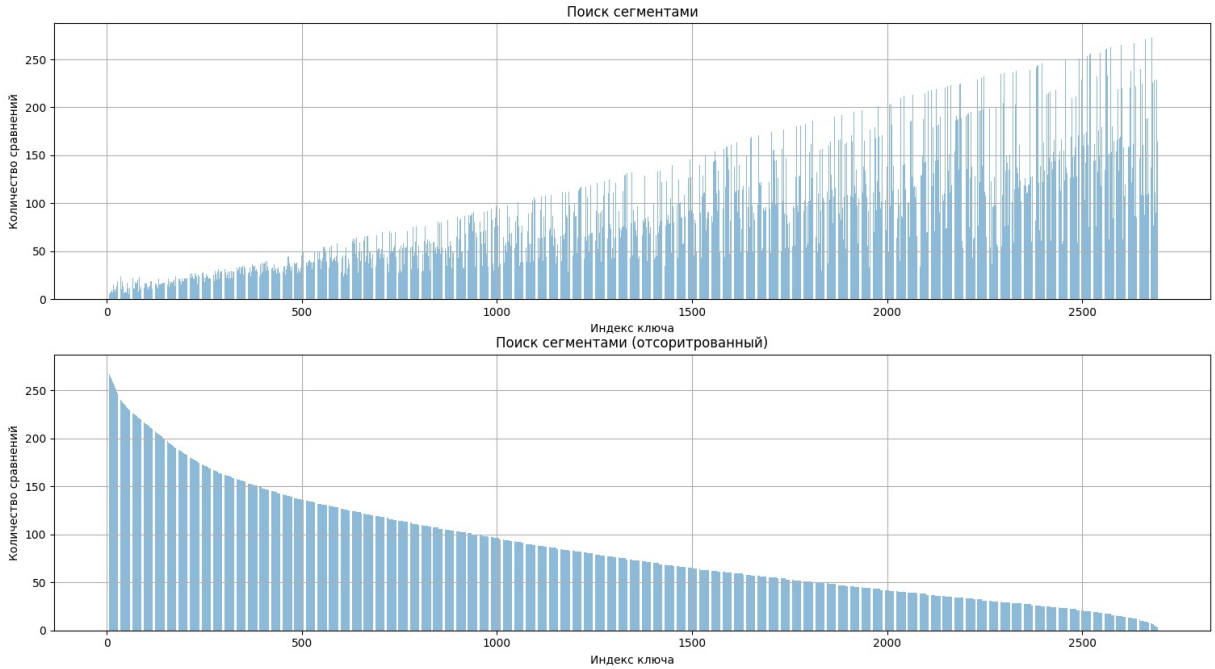

Рисунок 4.5 – Сравнение по количеству сравнений при поиске ключа в словаре алгоритмом разбиения на сегменты по общему признаку ключей

4.5 Вывод

В результате эксперимента было получено, что алгоритм поиска в словаре разбиением на сегменты является самым быстрым алгоритмом (при далеких от начала словаря ключах быстрее алгоритма полного перебора в сотни раз, а также примерно в 10 раз быстрее алгоритма бинарного поиска) и имеет примерно одинаковое время поиска каждого ключа, как и алгоритм бинарного поиска. При этом скорость алгоритма полного перебора зависит от положения ключа в словаре и увеличивается пропорционально дальности нахождения ключа от начала словаря. Следовательно, для быстрого поиска стоит выбирать алгоритм поиска с разбиением на сегменты, но стоит максимально аккуратно разбивать на сегменты, чтобы добиться максимальной производительности.

Также при проведении эксперимента было определено, что алгоритм бинарного поиска в среднем требует минимального количества сравнений, по сравнению с остальными алгоритмами. Для словаря, который использовался при выполнении данной текущей работы алгоритму бинарного поиска нужно максимум 12 сравнений, для алгоритма с разбиением на сегменты – 274 сравнения, а для алгоритма полного перебора – 2695 сравнений. То есть, при необходимости, когда нужно сократить количество сравнений, стоит использовать алгоритм бинарного поиска в словаре.

Заключение

В результате исследования было определено, что самым быстрым алгоритмом поиска является метод разбиения на сегменты, которые примерно в 10 раз быстрее бинарного поиска и в сотни раз быстрее алгоритма поиска полным перебором, время которого зависит от удаленности ключа в словаре. Также бинарный алгоритм является алгоритмом, которому для нахождения элемента в словаре нужно меньше всего сравнений. Так, в словаре, который используется в данной лабораторной работе, для поиска информации по ключу требуется максимум 12 сравнений (274 – для алгоритма разбиения на сегменты, 2695 сравнений – для алгоритма полного перебора).

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

- были изучены алгоритмы поиска информации по ключу в словаре – полным перебором, бинарного поиска и поиска с разбиением на сегменты;
- были реализованы алгоритмы полного перебора, бинарного поиска и поиска с разбиением на сегменты;
- проведен сравнительный анализ по времени алгоритмов на всех ключах словаря;
- проведена сравнительный анализ по количеству сравнений, необходимых алгоритму для нахождения каждого ключа в словаре;
- подготовлен отчёт о лабораторной работе.

Список литературы

[1] Словари [Электронный ресурс]. Режим доступа: https://younglinux.info/python/dictionary (дата обращения: 27.11.2021).

[2] Полный перебор [Электронный ресурс]. Режим доступа: http://skud-perm.ru/posts/polnyj-perebor (дата обращения: 27.11.2021).

[3] Бинарный поиск [Электронный ресурс]. Режим доступа: https://prog-cpp.ru/search-binary/ (дата обращения: 27.11.2021).

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

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

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

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

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

[9] 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).