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»

Лабораторная работа №7
по дисциплине "Анализ Алгоритмов"

Тема Поиск в словаре

Студент Ковалец К. Э.

Группа ИУ7-53Б

Преподаватель Волкова Л. Л.

Москва — 2021 г.

Содержание

Введение

Словарь – абстрактный тип данных, позволяющий хранить пары вида «ключ – значение» и поддерживающий операции добавления пары, а также поиска и удаления пары по ключу. В паре (k, v) значение v называется значением, ассоциированным с ключом k. Поиск – основная задача при использовании словаря. Данная задача решается различными способами, которые отличаются временем выполнения.

Целью данной лабораторной работы является изучение алгоритмов поиска в словаре – полным перебором, бинарным поиском и поиском сегментами.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- исследовать понятие словаря;
- изучить алгоритмы поиска в словаре (перебором, бинарным поиском, сегментами);
- привести схемы используемых алгоритмов;
- описать используемые структуры данных;
- описать структуру разрабатываемого ПО;
- определить средства программной реализации;
- реализовать разработанные алгоритмы;
- провести функциональное тестирование;
- провести сравнительный анализ времени работы алгоритмов;
- описать и обосновать полученные результаты в отчете о выполненной лабораторной работе.

1 Аналитическая часть

В данном разделе будет представлено описание понятия словаря и описаны алгоритмы поиска в нём (полным перебором, бинарным поиском, поиском сегментами).

1.1 Словарь

Словарь [1] – абстрактный тип данных, позволяющий хранить пары вида «ключ – значение» и поддерживающий операции добавления пары, а также поиска и удаления пары по ключу. В паре (k, v) значение v называется значением, ассоциированным с ключом k.

Поиск – основная задача при использовании словаря. Данная задача решается различными способами, которые отличаются временем выполнения.

В данной лабораторной работе использовался словарь со следующими параметрами:

- ключ - фамилия клиента;
- значение - информация о клиенте:
  - имя;
  - почта;
  - телефон.

1.2 Алгоритм поиска полным перебором

Поиск полным перебором [2] – метод решения, при котором поочередно перебираются все ключи словаря, пока не будет найден нужный.

Сложность такого алгоритма зависит от количества всех возможных решений, а время решения может стремиться к экспоненциальному времени работы. Чем дальше искомый ключ от начала словаря, тем выше трудоемкость алгоритма. Пусть на старте алгоритм затрагивает k_0 операций, а при сравнении k_1 операций, тогда:

- в лучшем случае элемент будет найден на первом сравнении за $k_0 + k_1$ операций;
- в худшем случае элемент будет найден на последнем сравнении за $k_0 + N \cdot k_1$ операций, где N – размер словаря;
- элемент будет найден на *i-ом* сравнении за $k_0 + i \cdot k_1$ операций;

Тогда средняя трудоемкость может быть рассчитана по следующей формуле:

$$f = k_0 + k_1 \cdot \left(1 + \frac{N}{2} - \frac{1}{N+1}\right) \tag{1.1}$$

1.3 Алгоритм бинарного поиска

Бинарный поиск [3] – поиск в заранее отсортированном словаре, который заключается в сравнении со средним элементом, и, если ключ меньше, то продолжать поиск в левой части тем же методом, иначе – в правой части.

Пусть на старте алгоритм затрагивает k_0 операций, тогда:

- в лучшем случае элемент будет найден на первом сравнении с средним элементом с трудоемкостью $k_0 + \log_2 1$;
- в худшем случае элемент будет найден на последнем сравнении с трудоёмкостью $b + \log_2 N$, где N – размер словаря;
- элемент будет найден на *i-ом* сравнении с трудоемкостью $b + \log_2 i$;

В случае большого объема данных и обратного порядка сортировки может произойти так, что алгоритм полного перебора будет эффективнее по времени, чем алгоритм двоичного поиска.

1.4 Алгоритм поиска сегментами

Поиск с помощью сегментов [4] – словарь разбивается на части, в каждую из которых попадают все элементы с некоторым общим признаком (для букв это может быть первая буква, для чисел – остаток от деления).

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 нём с помощью бинарного поиска ищется нужный ключ.

В лучшем случае первым сегментом будет выбран тот, серединный элемент которого окажется нужным.

В худшем случае последним сегментом будет выбран тот, который будет содержать нужный элемент, при этом сложность поиска ключа в данном сегменте – $\log_2 N$, где N - число элементов в сегменте.

Тогда средняя трудоемкость поиска i-го элемента может быть рассчитана по следующей формуле:

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

1.5 Вывод

В этом разделе были описаны понятие словаря и алгоритмы поиска в нём (полным перебором, бинарным поиском, поиском сегментами).

На вход программе будет поступать словарь (в нужной форме для каждого конкретного алгоритма), а также ключ, по которому будет происходить поиск в словаре. При попытке задать некорректные данные, будет выдано сообщение об ошибке. Если клинт с заданным ключом не будет найден, то выведется соответствующее сообщение. Реализуемое ПО будет давать возможность выбрать алгоритм поиска в словаре (полным перебором, бинарным поиском, поиском сегментами) и вывести для него результат вычисления, а также возможность произвести сравнение алгоритмов по затраченному времени и кол-ву используемых сравнений.

2 Конструкторская часть

В данном разделе будут приведены схемы алгоритмов поиска в словаре (полным перебором, бинарным поиском, поиском сегментами), приведено описание используемых типов данных, классов эквивалентности, а также описана структура ПО.

2.1 Схемы алгоритмов

На рис. 2.1 - 2.3 приведены схемы алгоритмов поиска в словаре (полным перебором, бинарным поиском, поиском сегментами).

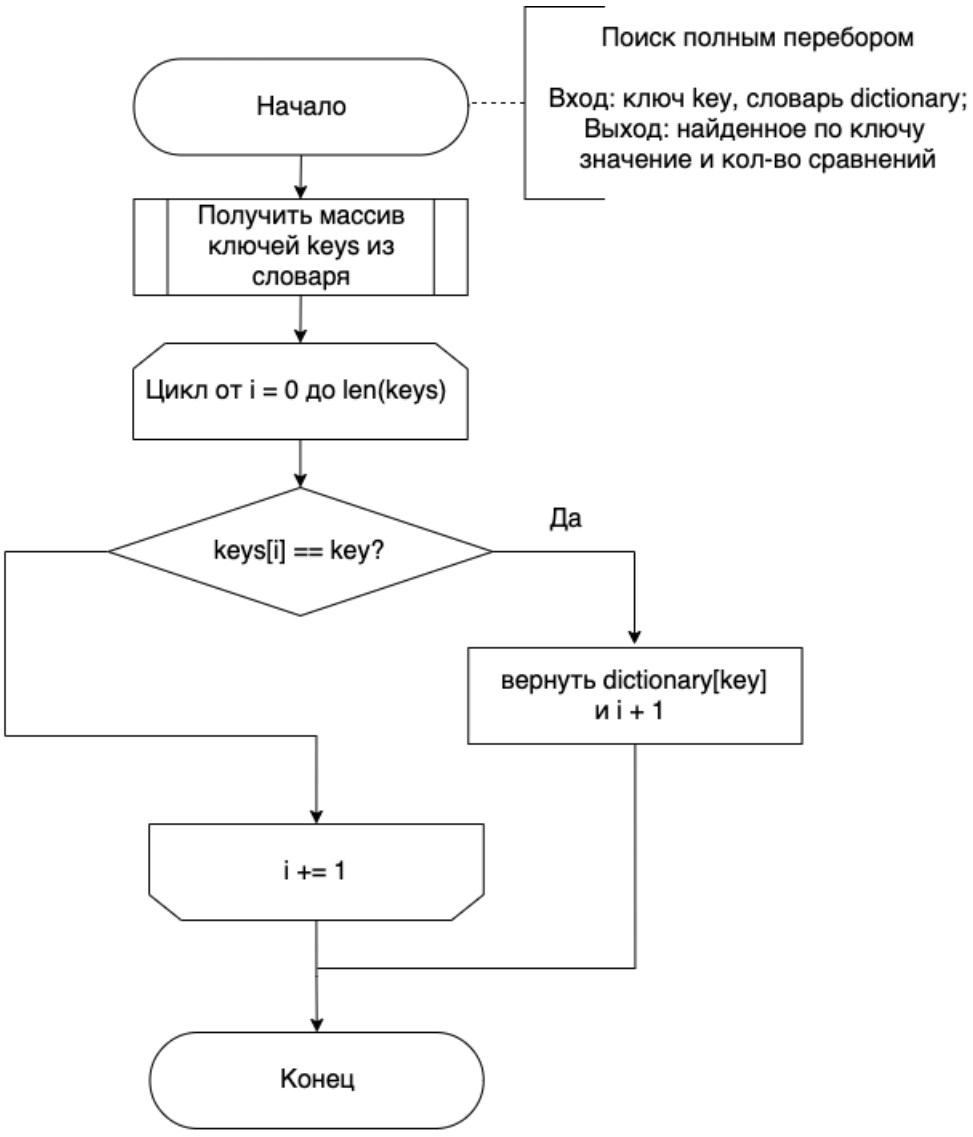

Рисунок 2.1 – Схема алгоритма поиска полным перебором

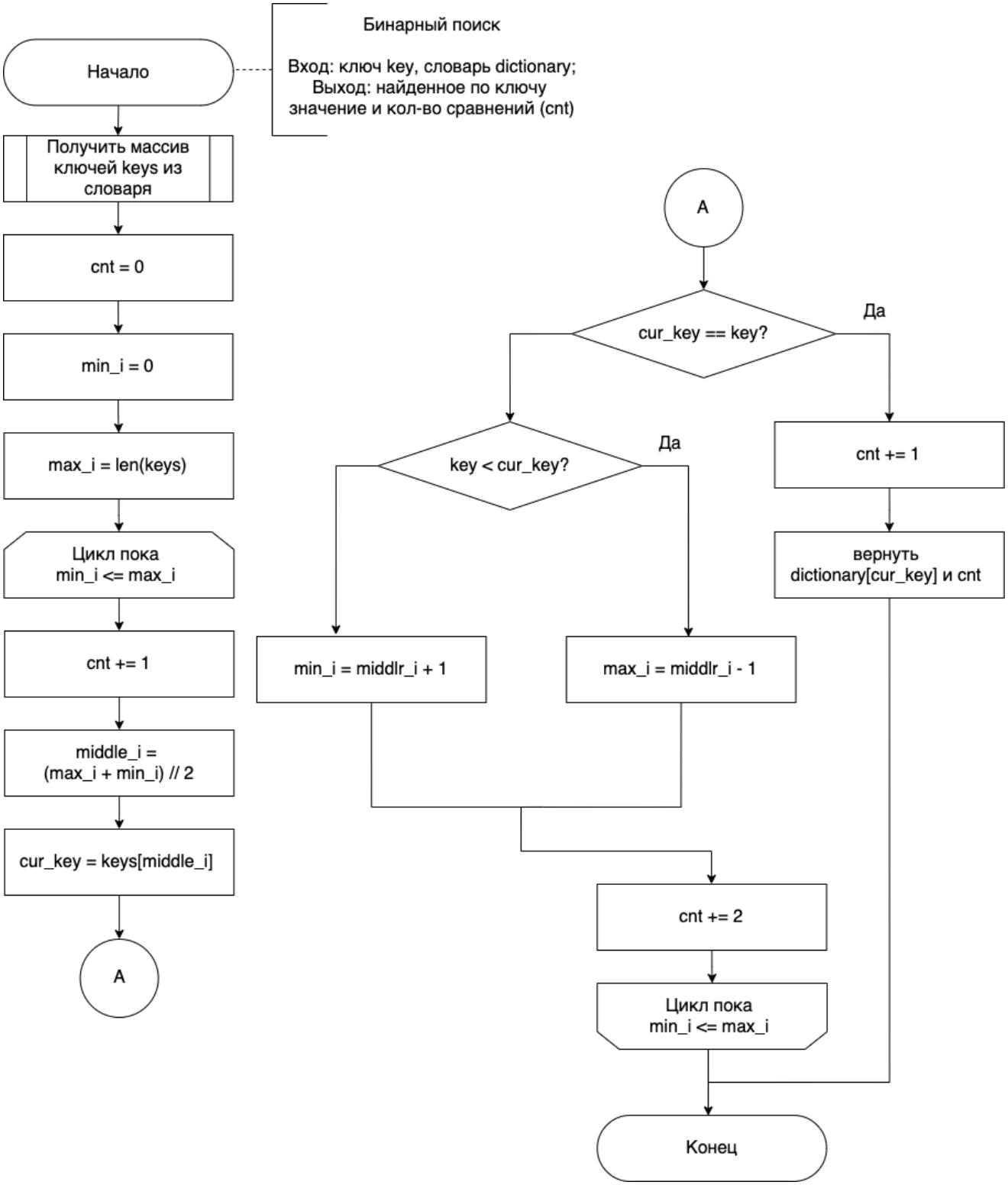

Рисунок 2.2 – Схема алгоритма бинарного поиска

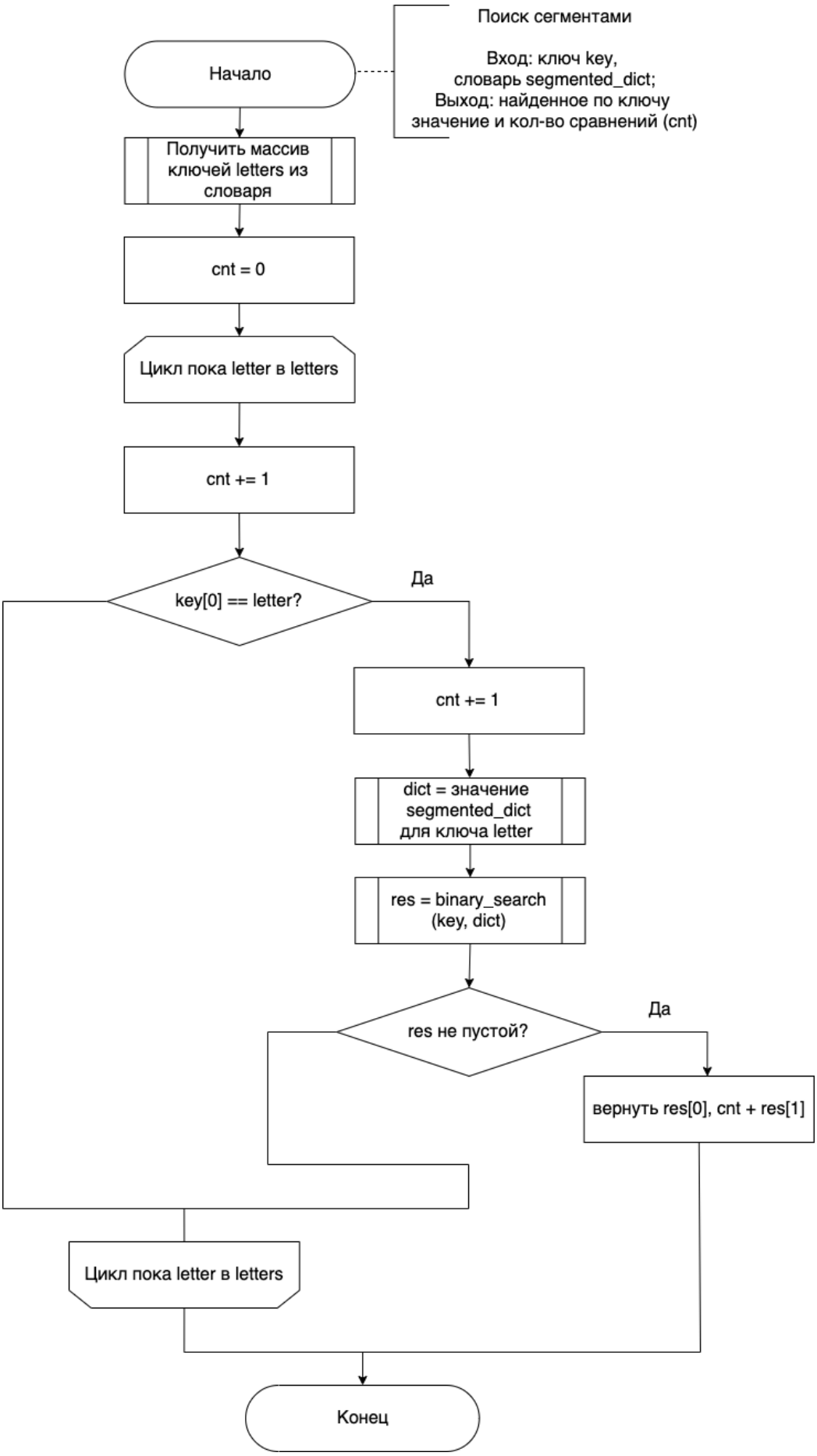

Рисунок 2.3 – Схема алгоритма поиска сегментами

2.2 Классы эквивалентности

Выделенные классы эквивалентности для тестирования:

- номер команды < 0 или > 5;
- номер команды не является целым числом;
- в словаре нет введённого ключа;
- в словаре есть введённый ключ.

2.3 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры данных:

- словарь - встроенный тип данных *dict* для словаря в *Python*;
- ключ в словаре - строка типа *str*;
- значение в словаре - строка типа *str*;
- название файла - строка типа *str*.

2.4 Структура ПО

Реализуемое ПО будет давать возможность выбрать алгоритм поиска в словаре (полным перебором, бинарным поиском, поиском сегментами) и вывести для него результат вычисления, а также возможность произвести сравнение алгоритмов по затраченному времени и кол-ву используемых сравнений. Для взаимодействия с пользователем будет разработано меню. В данном ПО будет реализован метод структурного программирования.

Для данного ПО будут разработаны следующие функции:

- функция для генерации файла с данными о клиентах, входные данные - имя создаваемого файла;
- функция для создание словаря, использующая информацию из файла, входные данные - имя файла, выходные - заполненный словарь;
- функции для реализации алгоритмов поиска значений в словаре по ключу (полным перебором, бинарным поиском, поиском сегментами), входные данные - ключ, словарь, выходные - значение в словере по заданному ключу, кол-во сравнений, использованных в данном алгоритме;
- функция для вывода найденного значения словаря по ключу, а также кол-ва используемых алгоритмом сравнений, входные данные - ключ, значение словаря, кол-во сравнений;
- функция создания сегментного словаря для использования алгоритма поиска сегментами, входные данные - словарь, выходные - сегментный словарь;
- функция сортировки элементов словаря по ключу, входные данные - словарь, выходные - отсортированный словарь;
- функция для построения гистограмм кол-ва сравнений, используемых каждым из алгоритмов;
- функция для построения графика зависимости времени работы алгоритмов от индекса ключа.

2.5 Вывод

В данном разделе на основе теоретических данных были построены схемы требуемых алгоритмов поиска в словаре (полным перебором, бинарным поиском, поиском сегментами), выбраны используемые типы данных, выделены классы эквивалентности для тестирования, а также была описана структура ПО.

3 Технологическая часть

В данном разделе будут приведены средства реализации, сведения о модулях программы, листинги кода, а также функциональные тесты.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования *Python* [5]. Выбор обсуловлен наличием опыта работы с ним. Время работы было замерено с помощью функции *process_time* из библиотеки *time* [6].

3.2 Сведения о модулях программы

ПО будет состоять из следующих модулей:

- *main.py* – файл, содержащий функцию *main*;
- *generate.py* – файл, содержащий функции для генирации файлов с данными о клиентах;
- *alg_search.py* – файл, в котором содержатся функции реализаций алгоритмов поиска значений в словаре по ключу;
- *dictionary.py* – файл, в котором содержатся функции для работы со словарём;
- *compare.py* – файл, в котором содержатся функции для сравнения колва сравнений внутри алгоритмов, замера времени работы алгоритмов и построения графика зависимоти времени выполнения от индекса ключа в словаре;
- *read.py* – файл, в котором содержатся функции ввода данных;
- *color.py* – файл, который содержит переменные типа *string* для цветного вывода результата работы программы в консоль.

3.3 Листинги кода

В листингах 3.1 - 3.3 представлены функции реализации алгоритмов поиска в словаре (полным перебором, бинарным поиском, поиском сегментами). В листинге 3.4 представлена функция создания словаря для использования поиска по нему сегментами.

Листинг 3.1 – Алгоритм поиска полным перебором

```
def full_search(key, dictionary):
    keys = list(dictionary.keys())

    for i in range(len(keys)):
        if keys[i] == key:
            return dictionary[key], i + 1
```

Листинг 3.2 – Алгоритм бинарного поиска

```
def binary_search(key, dictionary):
    keys = list(dictionary.keys())

    cnt = 0
    min_i = 0
    max_i = len(keys)

    while min_i <= max_i:
        cnt += 1

        middle_i = (max_i + min_i) // 2
        cur_key = keys[middle_i]

        if cur_key == key:
            cnt += 1

            return dictionary[cur_key], cnt

        elif key < cur_key:
            max_i = middle_i - 1
        else:
            min_i = middle_i + 1

        cnt += 2
```

Листинг 3.3 – Алгоритм поиска сегментами

```
def segment_search(key, segmented_dict):
    letters = list(segmented_dict.keys())

    cnt = 0

    for letter in letters:
        cnt += 1

        if key[0] == letter:
            cnt += 1
            res = binary_search(key, segmented_dict.get(letter))

            if res is not None:
                return res[0], cnt + res[1]
```

Листинг 3.4 – Функция создания словаря для использования поиска по нему сегментами

```
def segment_dict(dictionary):
    tmp_dict = {i: 0 for i in "ЙЦУКЕНГШЩЗХФЫВАПРОЛДЖЭЁЯЧСМИТБЮ"}

    for key in dictionary:
        tmp_dict[key[0]] += 1

    tmp_dict = sort_dict_by_data(tmp_dict)

    segmented_dict = tmp_dict

    for key in segmented_dict:
        segmented_dict[key] = dict()

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

    return segmented_dict
```

3.4 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для алгоритмов поиска в словаре (полным перебором, бинарным поиском, поиском сегментами). Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Входные данные	Алгоритм	Результат
команда 10		Сообщение об ошибке
команда k		Сообщение об ошибке
"Морозова"	Полный перебор	Такого клиента нет
"Морозова"	Бинарный поиск	Такого клиента нет
"Морозова"	Поиск сегментами	Такого клиента нет
"Петров"(случайный ключ)	Полный перебор	Информация о клиенте
"Петров"(случайный ключ)	Бинарный поиск	Информация о клиенте
"Петров"(случайный ключ)	Поиск сегментами	Информация о клиенте
"Аалферов"(первый ключ)	Полный перебор	Информация о клиенте
"Аалферов"(первый ключ)	Бинарный поиск	Информация о клиенте
"Аалферов"(первый ключ)	Поиск сегментами	Информация о клиенте
"Яяпмей"(последний ключ)	Полный перебор	Информация о клиенте
"Яяпмей"(последний ключ)	Бинарный поиск	Информация о клиенте
"Яяпмей"(последний ключ)	Поиск сегментами	Информация о клиенте

3.5 Вывод

В данном разделе были разработаны алгоритмы поиска в словаре (полным перебором, бинарным поиском, поиском сегментами), проведено тестирование, описаны средства реализации и сведения о модулях программы.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование представлены далее.

- Операционная система: macOS 11.5.2. [7]
- Память: 8 GiB.
- Процессор: 2,3 GHz 4-ядерный процессор Intel Core i5. [8]

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

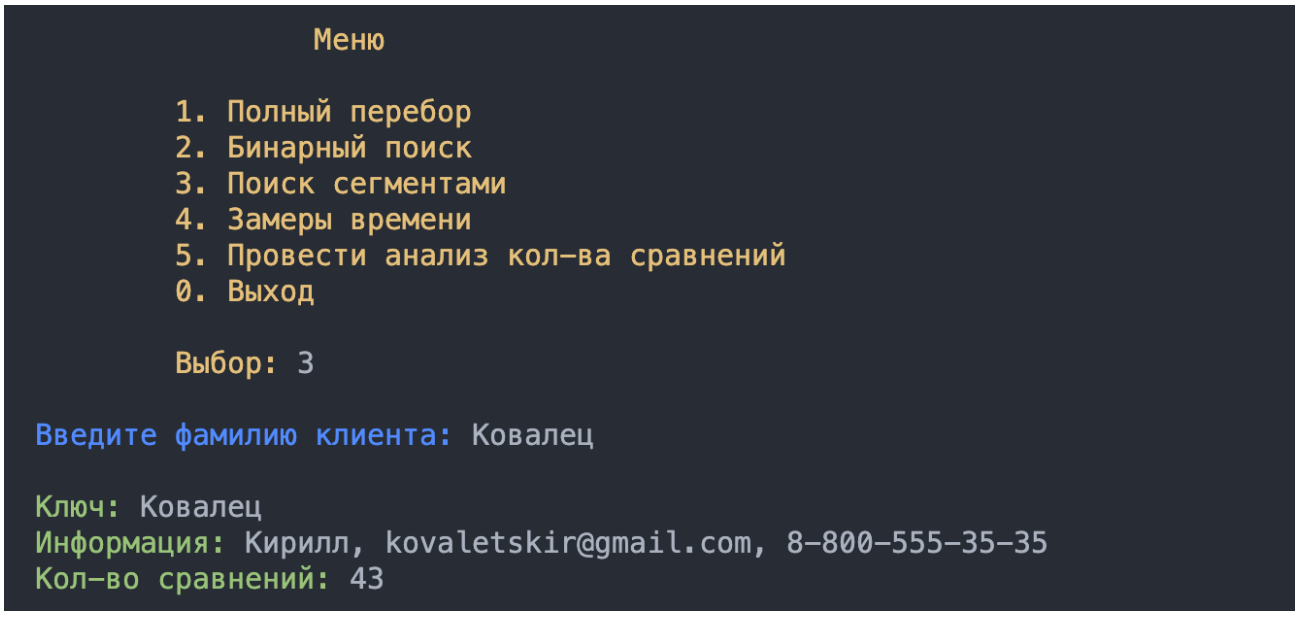

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения алгоритмов

Результаты замеров времени работы алгоритмов поиска в словаре (полным перебором, бинарным поиском, поиском сегментами) представлены на рисунках 4.2 - 4.3. Замеры времени проводились в секундах и усреднялись для каждого набора одинаковых экспериментов.

Инд. ключа	Полный перебор	Бинарный перебор	Поиск сегментами
0	3.2173e-05	2.5123e-05	4.6920e-06
500	4.8212e-05	2.7302e-05	5.6980e-06
1000	7.8103e-05	2.4966e-05	4.9320e-06
1500	1.0981e-04	2.3720e-05	4.7840e-06
2000	1.3031e-04	2.4688e-05	4.7910e-06
2500	1.5886e-04	2.4569e-05	4.0790e-06
3000	1.8669e-04	2.4104e-05	4.9060e-06

Рисунок 4.2 – Замеры времени работы алгоритмов поиска в словаре

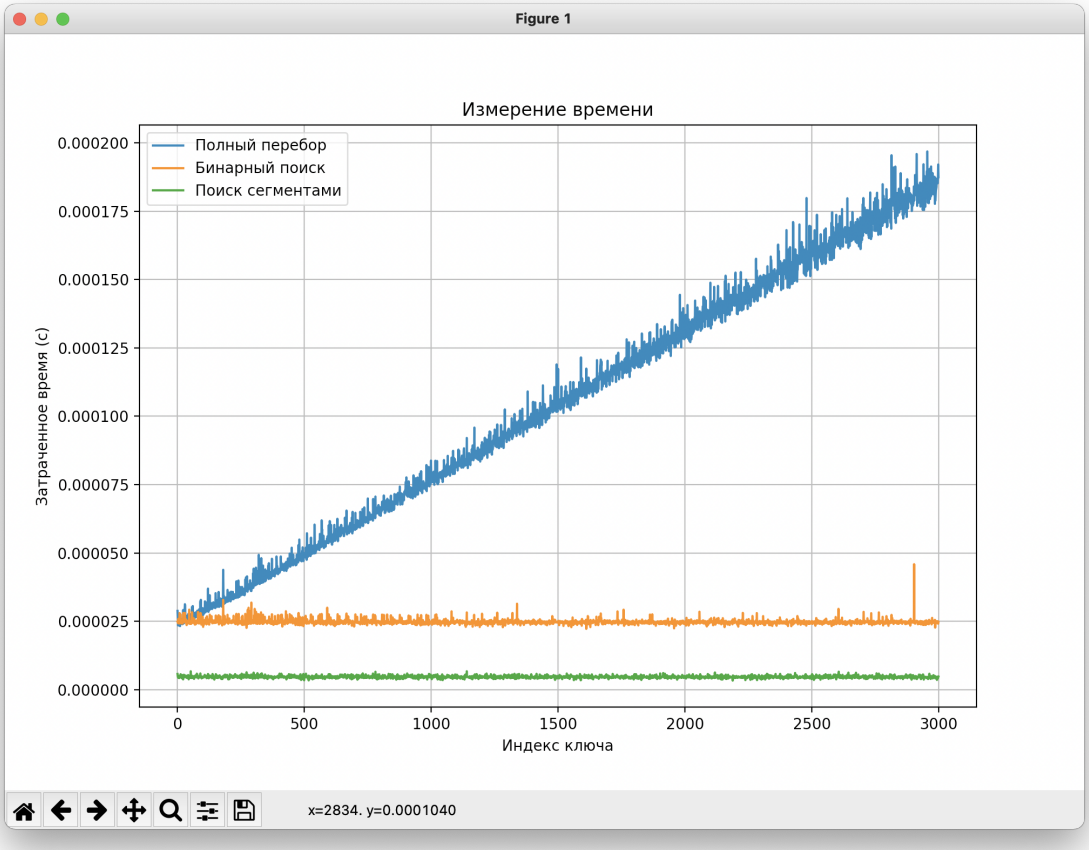

Рисунок 4.3 – Зависимость времени работы алгоритмов поиска от индекса ключа

4.4 Кол-во сравнений при работе алгоритмов

Для каждого алгоритма был проведён анализ по количеству сравнений для нахождения каждого ключа в словаре. Были составлены по две гистограммы для всех алгоритмов поиска (ключи расположены в том же порядке, как и в словаре, и когда ключи отсортировнны в порядке убывания).

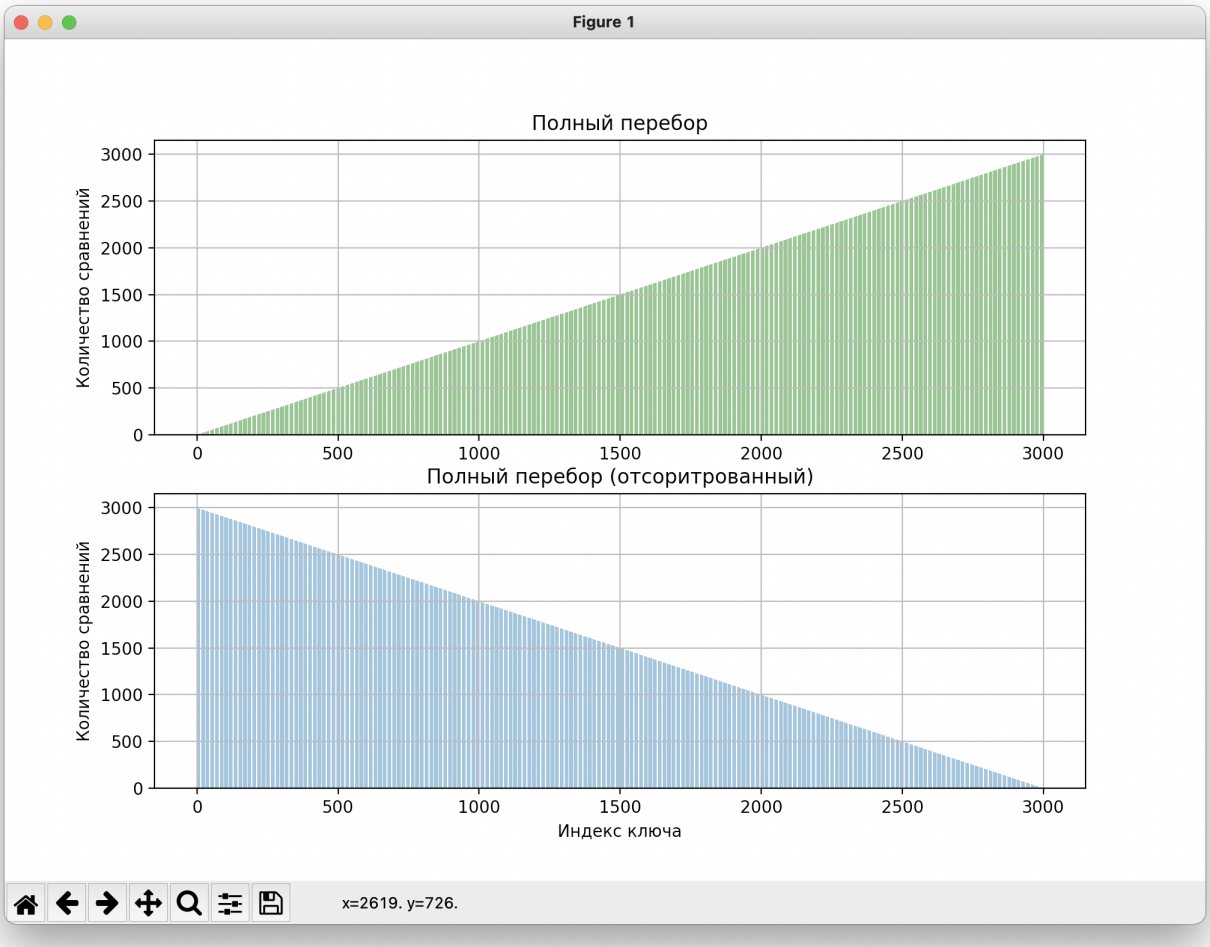

Рисунок 4.4 – Кол-во сравнений при поиске ключа в словаре (полным перебором)

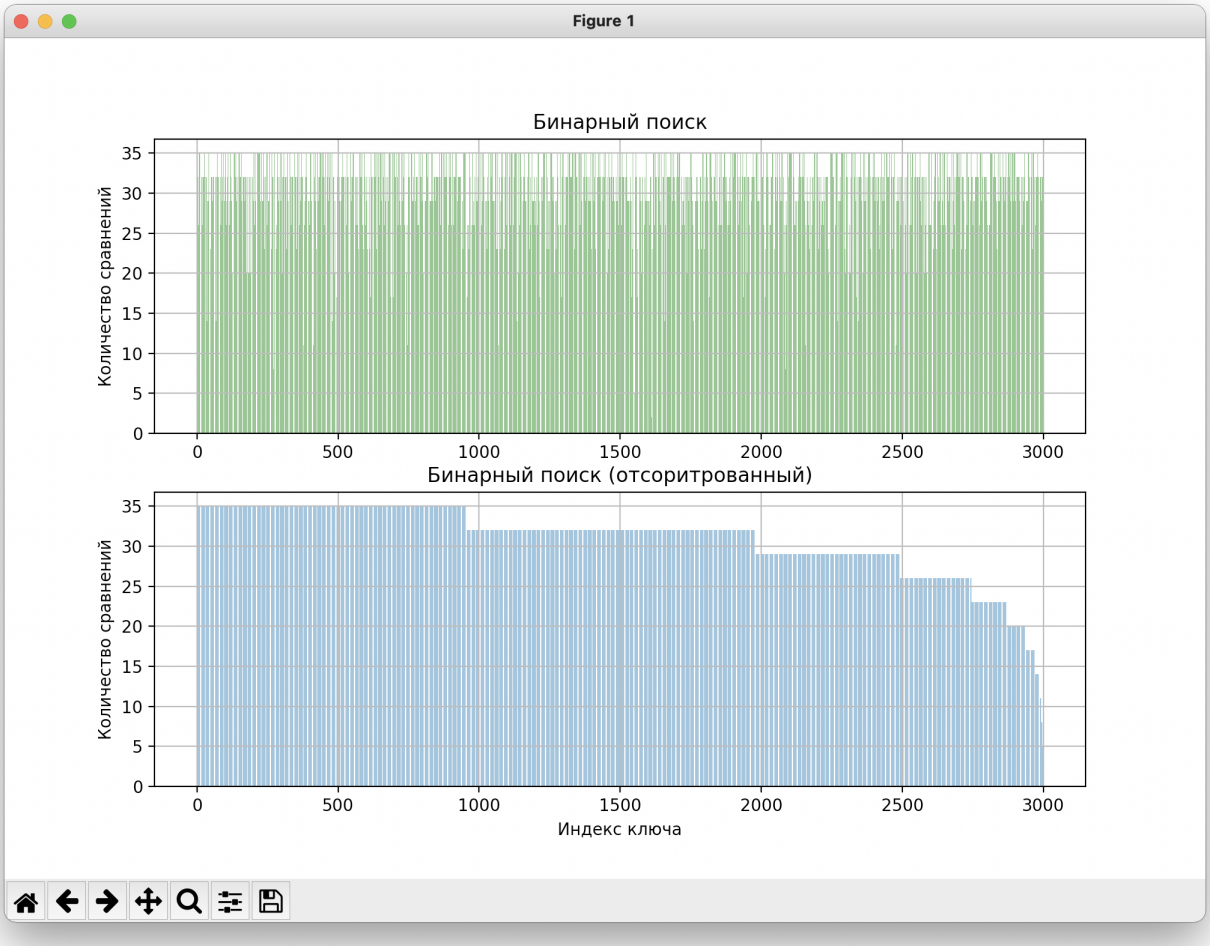

Рисунок 4.5 – Кол-во сравнений при поиске ключа в словаре (бинарным поиском)

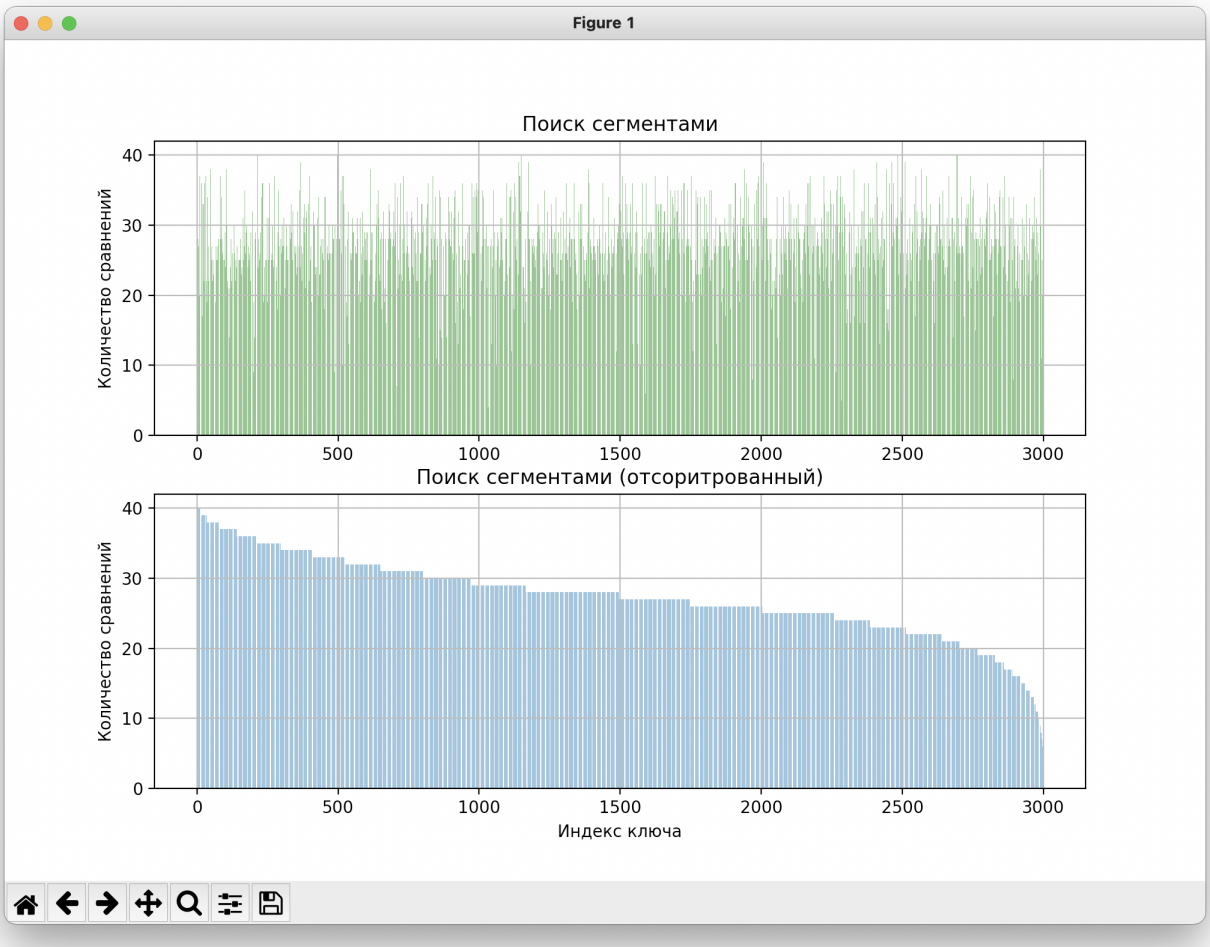

Рисунок 4.6 – Кол-во сравнений при поиске ключа в словаре (поиском сегментами)

4.5 Вывод

В этом разделе были указаны технические характеристики машины, на которой происходило сравнение времени работы алгоритмов поиска в словаре (полным перебором, бинарным поиском, поиском сегментами), а также кол-ва сравнений, используемых каждым из алгоритмов.

В результате замеров времени было установлено, что алгоритм поиска сегментами работает лучше остальных алгоритмов на всех ключах (В 5.1 раз быстрее алгоритма бинарного поиска при любом значении ключа и в 15.8 раз быстрее алгоритма поиска полным перебором на индексе ключа - 1000 и в 38.1 раз быстрее на индексе ключа - 3000). Стоит отметить, что алгоритм бинарного поиска и алгоритм поиска сегментами имеют примерно одинаковое время поиска каждого ключа, а скорость алгоритма полного перебора зависит от положения ключа в словаре (чем дальше ключ от начала словаря, тем дольше будет работать поиск).

Также в результате измерения кол-ва сравнений, используемых каждым из алгоритмов было установлено, что алгоритм поиска сегментами в среднем использует минимальное кол-во сравнений для нахождения значения ключа - 28 (алгоритм бинарного поиска использовал - 32 сравнения, а алгоритм поиска полным перебором - 1500). При этом максимальное кол-во сравнений для алгоритма поиска сегментами составило 40 сравнений, для алгоритма бинарного поиска - 35 сравнений, а для алгоритма поиска полным перебором - 3000 сравнений.

Заключение

Было экспериментально подтверждено различие во временной эффективности алгоритмов поиска в словаре (полным перебором, бинарным поиском, поиском сегментами). В результате исследований можно сделать вывод о том, что при возможности стоит использовать алгоритм поиска сегментами, так как он работает лучше остальных алгоритмов на всех ключах (В 5.1 раз быстрее алгоритма бинарного поиска при любом значении ключа и в 38.1 раз быстрее алгоритма поиска полным перебором на индексе ключа - 3000), а так же он в среднем использует минимальное кол-во сравнений (28). Однако стоит помнить, что для алгоритма поиска с помощью сегментов требуется разбить словарь на части (сегменты), что может быть не всегда удобно.

В ходе выполнения данной лабораторной работы были решены следующие задачи:

- изучены алгоритмы поиска в словаре (полным перебором, бинарным поиском, поиском сегментами);
- применены изученные основы для реализации описанных алгоритмов;
- произведен сравнительный анализ алгоритмов поиска в словаре;
- экспериментально подтверждено различие во временной эффективности рассматриваемых алгоритмов при помощи разработанного программного обеспечения на материале замеров процессорного времени;
- произведен сравнительный анализ по количеству сравнений, необходимых алгоритмам для нахождения каждого ключа в словаре;
- описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе.

Поставленная цель была достигнута.

Литература

[1] Словари [Электронный ресурс]. Режим доступа: https://younglinux.info/python/dictionary (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: 18.10.2021).

[6] time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html#functions (дата обращения: 18.10.2021).

[7] macOS Monterey - Apple(RU) [Электронный ресурс]. Режим доступа: https://www.apple.com/ru/macos/monterey/ (дата обращения: 14.10.2021).

[8] Intel [Электронный ресурс]. Режим доступа: https://www.intel.ru/content/www/ru/ru/products/details/processors/core/i5.html (дата обращения: 14.10.2021).