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 Прянишников А. Н.

Группа ИУ7-55Б

Оценка (баллы) _____

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Содержание

Введение

Словарь - структура данных, позволяющая идентифицировать её элементы не по числовому индексу, а по произвольному. В жизни встречаются книжные бумажные словари и телефонные справочники - они являются словарями. Особенностью словаря является его динамичность: можно добавлять новые элементы с произвольными ключами и удалять уже существующие элементы.

Поиск в словаре – одна из основных операций над структурой данных, и перебор ключей не всегда отвечает требованиям к скорости работы программы. Поэтому существует несколько оптимизаций поиска в словаре.

Цель лабораторной работы – разработать ПО, которое реализует три алгоритма поиска в словаре: перебор ключей, бинарный поиск и поиск с использованием сегментации.

Для достижения поставленной цели необходимо выполнить следующее:

- рассмотреть понятие словаря;
- рассмотреть три основных алгоритма поиска в словаре;
- привести схемы реализации алгоритмов;
- определить средства программной реализации;
- реализовать три алгоритма для решения задачи;
- протестировать разработанное ПО;
- оценить реализацию алгоритмов по времени и памяти.

1 Аналитический раздел

В этом разделе будут рассмотрены понятие словаря, три основных алгоритма поиска в словаре, а также будут приведены требования к ПО.

1.1 Словарь

Словарь [1] – структура данных, который позволяет хранить пары вида "ключ-значение" – *(k, v)*. Словарь поддерживает три операции:

1. Добавление новой пары в словарь.
2. Поиск по ключу в словаре.
3. Удаление по ключу в словаре.

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.

1.2 Алгоритм полного перебора

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

Чем дальше искомый ключ от начала словаря, тем выше трудоемкость алгоритм. Пусть для поиска потребовалось k сравнений, а весь объём словаря – N. Тогда средняя трудоемкость равна:

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

1.3 Бинарный поиск

Бинарный поиск [3] – поиск в заранее отсортированном словаре, который заключается в сравнении со средним элементом, и, если ключ мень-

ше, то продолжать поиск в левой части тем же методом, иначе – в правой части.

Алгоритм работает только на отсортированном словаре, поэтому предварительно требуется сделать сортировку ключей в правильном порядке (например, лексикографическом).

Пусть k – количество сравнений, требуемых для поиска слова. Тогда трудоёмкость алгоритма можно представить так:

$$f = b + \log_2 k \tag{1.2}$$

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

$$P_i = \sum_j p_j \tag{1.3}$$

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.

1.5 Реализация словаря

Для реализации была выбрана база водителей из курса "Базы Данных". Ключом выступает номер паспорта, а значением - ФИО водителей. Гарантируется то, что номер паспорта уникальный. Общее количество – 20000 записей.

1.6 Требования к ПО

Ниже будет представлен список требований к разрабатываемому программному обеспечению.

Требования к входным данным:

- на вход подаётся ключ для поиска – номер паспорта, состоящий из 10 цифр(без пробелов);
- на вход подаётся алгоритм поиска в словаре.

Требования к выводу:

- программа должна вывести ФИО, соответствующее введённому номеру паспорта;
- если вводного ключа нет в словаре, то программа должна вывести соответствующее информационное сообщение.

Также программа должна предоставлять возможность измерить время поиска для каждого из алгоритмов.

1.7 Вывод

Были рассмотрены понятие словаря, три основных алгоритма поиска в словаре, а также были приведены требования к ПО.

2 Конструкторский раздел

В этом разделе будут приведены схемы реализации алгоритмов, а также выбранные классы эквивалентности для тестирования ПО.

2.1 Схемы алгоритмов

На рисунке 2.1 будет приведена схема реализации алгоритма полного перебора.

На рисунке 2.2 будет приведена схема реализации алгоритма бинарного поиска.

На рисунке 2.3 будет приведена схема реализации алгоритма поиска с использованием сегментов.

Рисунок. 2.1: Схема реализации алгоритма полного перебора

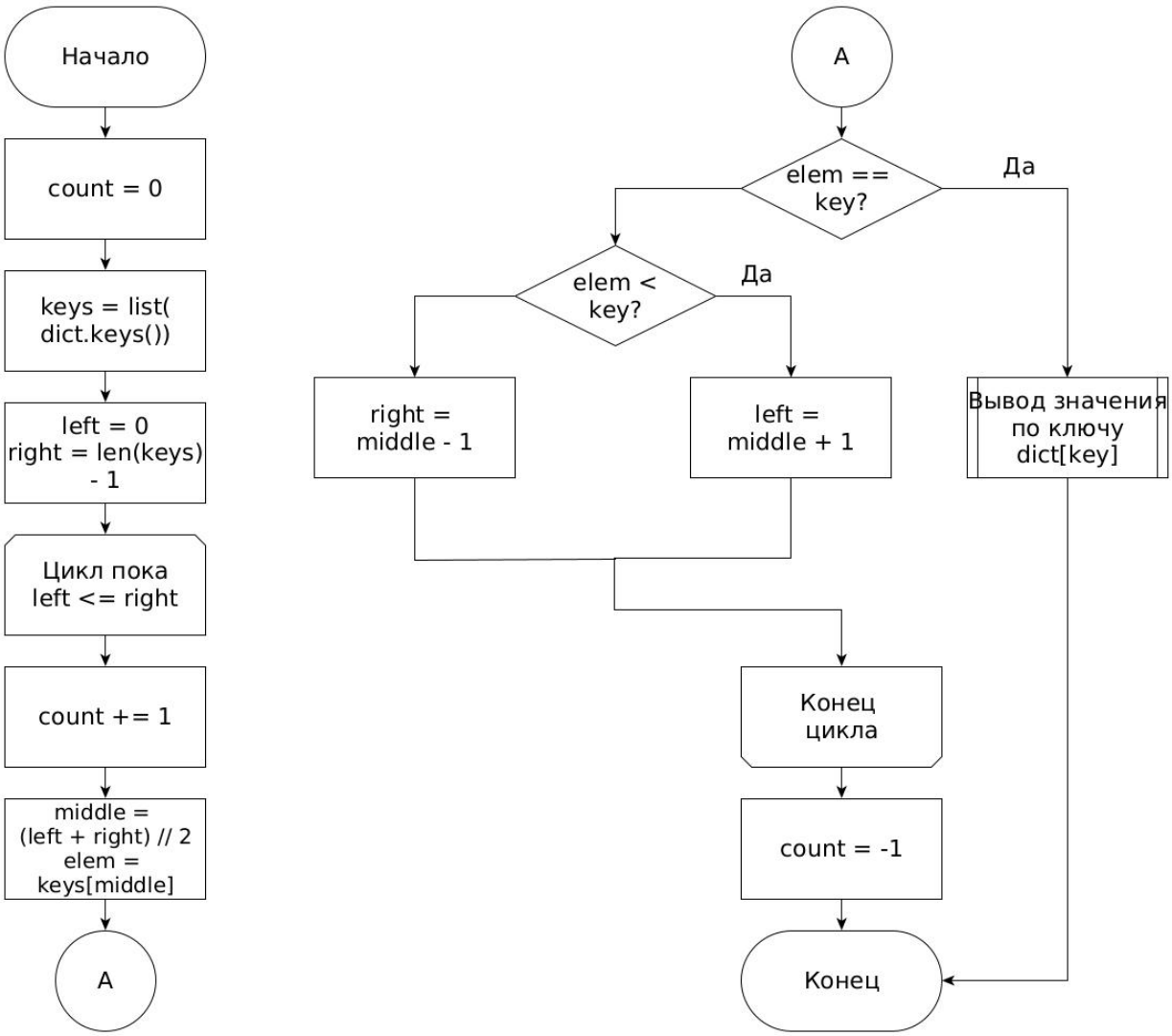

Рисунок. 2.2: Схема реализации алгоритма бинарного поиска

Рисунок. 2.3: Схема реализации поиска с использованием сегмента

2.2 Типы данных для алгоритмов

Для реализации алгоритма будет использоваться тип данных dict из python. Ключ – номер паспорта, является строковой переменной. Значение ключа – список, состоящий из трёх строковых переменных, соответствующих ФИО.

2.3 Способ тестирования

Тестирование программы будет производиться методом чёрного ящика. Такой подход выбран, так как от реализаций алгоритмов требуется в первую очередь правильность работы. Сама по себе реализация не требует тестировки, так как в точности повторяет теоретические принципы, сформированные в аналитическом разделе.

В качестве классов эквивалентности были выбраны следующие сущности:

- ключ, которого нет в словаре;
- ключ, который есть в словаре;
- ключ, который является первым в словаре;
- ключ, который является последним в словаре.

2.4 Вывод

Были приведены схемы реализации алгоритмов и типы данных для алгоритмов. Был определен способ тестирования алгоритмов.

3 Технологическая часть

В этом разделе будут приведены листинги кода и результаты функционального тестирования.

3.1 Средства реализации программного обеспечения

В качестве языка программирования выбран Python 3.9, так как имеется опыт разработки проектов на этом языке. Для замера процессорного времени используется функция process_time_ns из библиотеки time. В её результат не включается время, когда процессор не выполняет задачу [5].

3.2 Листинг кода

На листинге 3.1 представлена реализация алгоритма полного перебора. На листинге 3.2 представлена реализация алгоритма бинарного поиска. На листинге 3.3 представлена реализация алгоритма поиска с помощью сегментов.

Листинг 3.1: Реализация алгоритма полного перебора

```
def full_search(dict, find_key, output = True):
    count = 0
    result = -1
    for key in dict.keys():
        if key == find_key:
            result = 0
            if (output):
                print_value(dict, count)
            break
        else:
            count += 1
    if (result == -1):
        count = -1
    return count
```

Листинг 3.2: Реализация алгоритма бинарного поиска

```
def sort_dict(my_dict):
    keys = list(my_dict.keys())
    keys.sort()

    tmp_dict = dict()

    for key in keys:
        tmp_dict[key] = my_dict[key]

    return tmp_dict

def binary_search(sort_dict, find_key, output = True):
    count = 0
    result = -1

    keys = list(sort_dict.keys())

    left = 0
    right = len(keys) - 1
    middle = len(keys) // 2
    while (left <= right):
        count += 1

        key = keys[middle]
        if (key == find_key):
            if (output):
                print_value(sort_dict, middle)
            result = 0
            break

        elif (key < find_key):
            left = middle + 1
        else:
            right = middle - 1
        middle = (left + right) // 2
    if (result == -1):
        count = -1
    return count
```

Листинг 3.3: Реализация алгоритма поиска с помощью сегментов

```
def sort_value(my_dict):
    sorted_dict = dict()

    items = list(my_dict.items())
    items.sort(key = lambda k: k[1], reverse = True)

    for elem in items:
        sorted_dict[elem[0]] = elem[1]
```

```
    return sorted_dict

def make_segments(my_dict):
    temp_dict = {i: 0 for i in "0123456789"}

    for key in my_dict:
        temp_dict[key[0]] += 1

    # Получаем словарь, в котором самые часто встречаемые первые буквы клю
       ча идут по убыванию
    temp_dict = sort_value(temp_dict)

    # В том же порядке создаём ключ
    segmented_dict = {i: dict() for i in temp_dict}

    # Проходим по словарю и добавляем в новый словарь по ключу - первая бу
       ква
    for key in my_dict:
        segmented_dict[key[0]].update({key: my_dict[key]})

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

    return segmented_dict


def segment_search(segmented_dict, key, output = True):
    count = 0
    keys = list(segmented_dict.keys())
    for key_letter in keys:
        count += 1
        if (key[0] == key_letter):
            count_search = binary_search(segmented_dict[key_letter], key,
               output)
            if (count_search == -1):
                count = -1
            else:
                count += count_search
            break
    return count
```

3.3 Функциональные тесты

В таблице 3.1 приведены результаты функциональных тестов. В первом столбце – ключ. Во втором столбце – результат.

Таблица 3.1: Функциональные тесты

Ключ	Результат
1234567890	Ключ не был обнаружен
6762048930	Носов Ипат Бориславович
0000023802	Доронин Богдан Ильясович
9999867026	Симонов Силантий Якубович

Все тесты алгоритмами были пройдены успешно.

3.4 Вывод

Были приведены листинги кода. Были проведены функциональные тесты. Все алгоритмы справились с тестированием.

4 Исследовательская часть

В этом разделе будет продемонстрирована работа программы, а также приведены результаты тестирования алгоритмов.

4.1 Демонстрация работы программы

На рисунке 4.1 приведена демонстрация работы программы.

```
--------------------
Меню работы со словарём: номер паспорта - ФИО человека
1 - поиск по ключу полным перебором
2 - поиск по ключу бинарным поиском
3 - поиск по ключу с помощью сегментов
4 - сравнение по поиску
5 - вывести первые 20 элементов словаря (всего - 20000)
0 - выход

Выбор: 3
Введите ключ для поиска: 9999867026
--------------------
Найден человек с таким паспортом:
('9999867026', ['Симонов', 'Силантий', 'Якубович'])
--------------------
```

Рисунок. 4.1: Демонстрация программы

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование, следующие:

- операционная система: Ubuntu 20.04.1 LTS;
- память: 8 GB;
- процессор: Intel Core i5-1135G7 @ 2.40GHz [6].
- количество ядер процессора: 8

Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.3 Тестирование программы

Тестирование будет производиться следующим образом. Программа проводит поиск по каждому ключу всеми тремя алгоритмами 30 раз и усредняет полученные времена. Затем строится график по каждому алгоритму.

Время измерялось в наносекундах. Во время не включалось создание сортированного массива для бинарного поиска и создание специального сегментированного словаря для третьего алгоритма. На рисунке 4.2 представлены результаты тестирования алгоритмов.

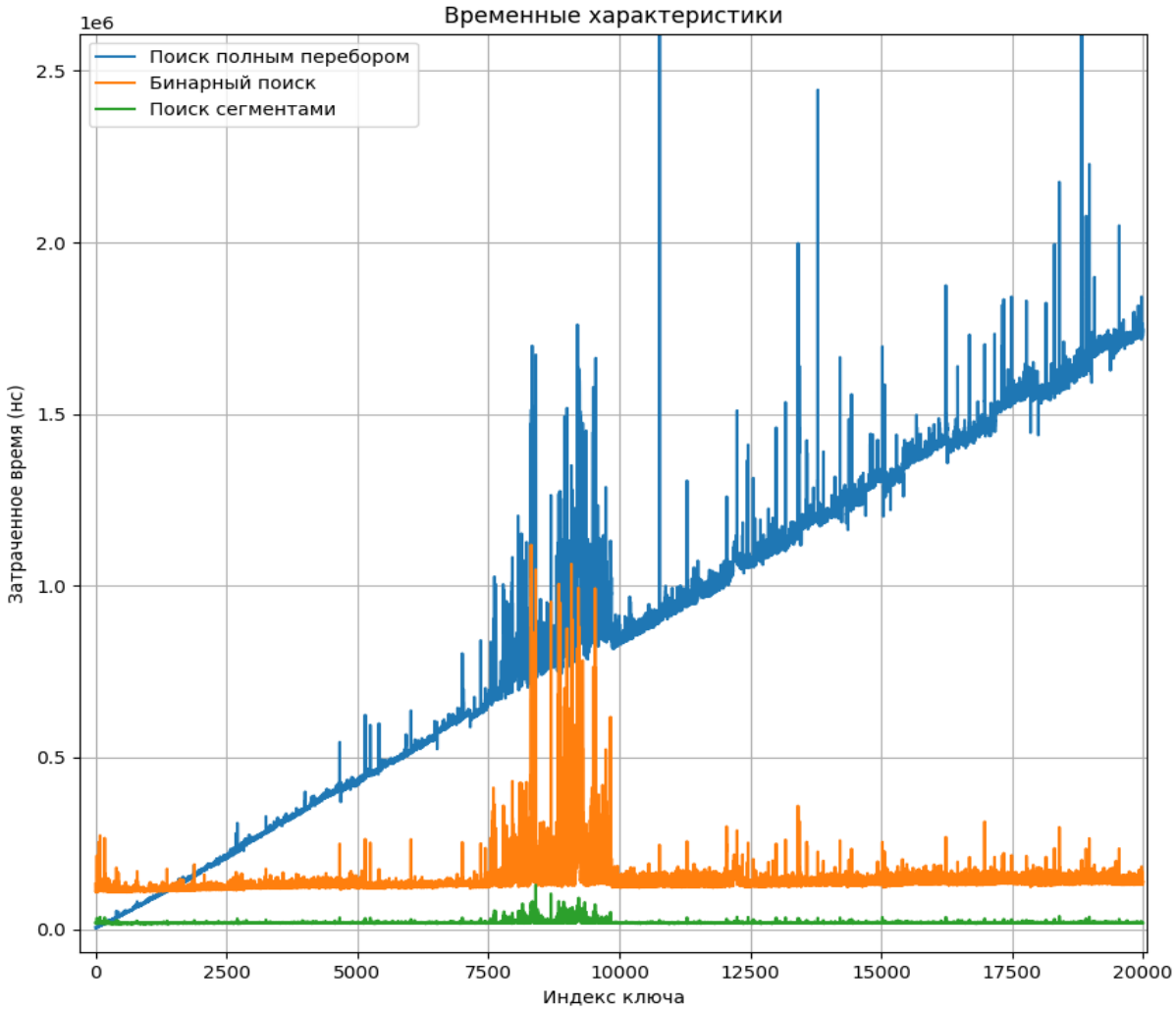

Рисунок. 4.2: Демонстрация программы

Перебор показал более стремительный рост по сравнению с остальными алгоритмами. На 20000-м ключе перебор оказался в семь раз медленнее, чем бинарный поиск, и в 70 раз медленнее, чем поиск с использованием сегментов. На первой тысяче элементов бинарный поиск оказался самым медленным. При этом время алгоритмов бинарного поиска и поиска с использованием сегментов практически не менялось в зависимости от номера ключа.

Также проведём тестирование алгоритмов по количеству операций сравнения в зависимости от ключа. Результаты тестирования приведены в таблице 4.1

Таблица 4.1: Результаты тестов

Ключ	Перебор	Бин.Поиск	Сегмент
6762048930	19984	15	11
0000023802	7405	14	20
9999867026	18700	15	17
9039639329	19964	12	16
8528870899	4003	15	13
8635632402	8058	14	13
1415792047	2048	15	10
7755312493	19981	13	16
0732110318	1	14	19
0506113949	19885	15	19

Видно, что для перебора количество сравнений зависит исключительно от того, каким по счёту ключ был добавлен в словарь. Перебор оказался самым эффективным алгоритмом только для самых первых ключей в словаре. У бинарного поиска и поиска с использованием сегментов количество сравнений не больше 20 для всех ключей. Это не удивительно, так как последний алгоритм использует внутри себя тот же самый бинарный поиск.

Время у поиска с использованием сегментов на такое же количество операций уходит меньше времени, так как поиск проводится на существенно меньшем количестве элементов.

4.4 Вывод

Была продемонстрирована работа программы, и приведены результаты тестирования алгоритмов. Поиск с использованием сегментов показал самое быстрое время практически на всех измерениях. Обычный перебор ключей на 20000-м ключе оказался в семь раз медленнее, чем бинарный поиск, и в 70 раз медленнее, чем поиск с использованием сегментов.

5 Заключение

В ходе работы было рассмотрено понятие словаря и рассмотрены три алгоритма поиска в словаре: полный перебор, бинарный поиск и поиск с использованием сегментов. Были составлены схемы алгоритмов. Было реализовано работоспособное ПО, удалось провести анализ зависимости затрат по времени от алгоритма.

Применимость алгоритмов зависит от того, насколько велик размер словаря. Алгоритм полного перебора перестаёт оправдывать себя после 1000-го ключа, а на 20000-м ключе он уступает по времени бинарному поиску в 7 раз, а поиску с использованием сегментов – в 70 раз.

Алгоритм бинарного поиска по времени оказался самым медленным при номере ключа до 1000. Алгоритм поиска с использованием сегментов оказался самым быстрым во всех случаях. При этом оба алгоритма работают за примерно одинаковое время при любом номере ключа. Количество сравнений для этих алгоритмов не больше 20.

Литература

[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] time - Time access and conversions. Режим доступа: https://docs.python.org/3/library/time.html (дата обращения: 21.09.2021).

[6] Процессор Intel® Core™ i5-1135G7. Режим доступа https://www.intel.ru/content/www/ru/ru/products/sku/208658/intel-core-i51135g7-processor-8m-cache-up-to-4-20-ghz/specifications.html (дата обращения: 15.10.2021).