



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 Динь Вьет Ань

Группа ИУ7И-54Б

Оценка (баллы)

Преподаватель Волкова Л.Л.

Содержание

Введение	3
1 Аналитическая часть	4
1.1 Словарь как структура данных	4
1.2 Алгоритм полного перебора	5
1.3 Алгоритм поиска в упорядоченном словаре двоичным поиском	6
1.4 Вывод	6
2 Конструкторская часть	7
2.1 Описание используемых типов данных	7
2.2 Структура разрабатываемого ПО	7
2.3 Схемы алгоритмов	7
2.4 Классы эквивалентности при тестировании	10
2.5 Вывод	10
3 Технологическая часть	11
3.1 Требования к ПО	11
3.2 Средства реализации	11
3.3 Реализация алгоритмов	11
3.4 Функциональное тестирование	12
3.5 Вывод	13
4 Исследовательская часть	14
4.1 Технические характеристики	14
4.2 Демонстрация работы программы	14
4.3 Время выполнения реализаций алгоритмов поиска ключа . .	15
4.4 Количество сравнений	16
4.5 Вывод	19
Заключение	20
Список литературы	21

Введение

Словарь, как тип данных, применяется везде, где есть связь “ключ – значение” или “объект – данные”: поиск налогов по ИНН и другое. Поиск — основная задача при использовании словаря. Данная задача решается различными способами, которые дают различную скорость решения.

Со временем стали разрабатывать алгоритмы поиска в словаре. В данной лабораторной работе мы рассмотрим два алгоритма.

1. Поиск полным перебором.
2. Бинарный поиск.

Цель данной работы — получить навык работы со словарём, как структурой данных, реализовать алгоритмы поиска по словарю (указаны выше) с применением оптимизаций.

Для достижения цели поставлены следующие задачи.

1. Изучить два алгоритма поиска в словаре.
2. Привести схемы алгоритмов поиска в словаре.
3. Описать структуру разрабатываемого программного обеспечения.
4. Применить изученные основы для реализации поиска значений в словаре по ключу.
5. Провести функциональное тестирование реализации разработанного алгоритма.
6. Получить замеры количества сравнений для каждого ключа (для всех двух алгоритмов).
7. Провести сравнительный анализ по времени для реализованных алгоритмов.
8. Подготовить отчет по лабораторной работе.

1 Аналитическая часть

В данном разделе описаны определение словаря как структуры данных, а также алгоритмы поиска по словарю.

1.1 Словарь как структура данных

Словарь (или "*ассоциативный массив*") [1] - абстрактный тип данных (интерфейс к хранилищу данных), позволяющий хранить пары вида «(ключ, значение)» и поддерживающий операции добавления пары, а также поиска и удаления пары по ключу:

- `INSERT(k, v);`
- `FIND(k);`
- `REMOVE(k).`

В паре (k, v) : v называется значением, ассоциированным с ключом k . Где k — это ключ, а v — значение. Семантика и названия вышеупомянутых операций в разных реализациях ассоциативного массива могут отличаться.

Операция `ПОИСК(k)` возвращает значение, ассоциированное с заданным ключом, или некоторый специальный объект `НЕ_НАЙДЕНО`, означающий, что значения, ассоциированного с заданным ключом, нет.

Ассоциативный массив с точки зрения интерфейса удобно рассматривать как обычный массив, в котором в качестве индексов можно использовать не только целые числа, но и значения других типов — например, строки.

В данной лабораторной работе в качестве ключа будет использоваться строка: название города, а в качестве значения — число: население этого города.

1.2 Алгоритм полного перебора

Алгоритмом полного перебора [2] называют метод решения задачи, при котором по очереди рассматриваются все возможные варианты. В случае реализации алгоритма в рамках данной работы будут последовательно перебираться ключи словаря до тех пор, пока не будет найден нужный.

Трудоёмкость алгоритма зависит от того, присутствует ли искомый ключ в словаре, и, если присутствует – насколько он далеко от начала массива ключей.

Пусть на старте алгоритм затрагивает k_0 операций, а при сравнении k_1 операций.

Пусть алгоритм нашёл элемент на первом сравнении (лучший случай), тогда будет затрачено $k_0 + k_1$ операций, на втором – $k_0 + 2 \cdot k_1$, на последнем (худший случай) – $k_0 + N \cdot k_1$. Если ключа нет в массиве ключей, то мы сможем понять это, только перебрав все ключи, таким образом трудоёмкость такого случая равно трудоёмкости случая с ключом на последней позиции. Трудоёмкость в среднем может быть рассчитана как математическое ожидание по формуле (1.1), где Ω – множество всех возможных случаев.

$$\begin{aligned} \sum_{i \in \Omega} p_i \cdot f_i &= (k_0 + k_1) \cdot \frac{1}{N+1} + (k_0 + 2 \cdot k_1) \cdot \frac{1}{N+1} + \\ &+ (k_0 + 3 \cdot k_1) \cdot \frac{1}{N+1} + (k_0 + N k_1) \frac{1}{N+1} + (k_0 + N \cdot k_1) \cdot \frac{1}{N+1} = \\ &= k_0 \frac{N+1}{N+1} + k_1 + \frac{1+2+\dots+N+N}{N+1} = \\ &= k_0 + k_1 \cdot \left(\frac{N}{N+1} + \frac{N}{2} \right) = k_0 + k_1 \cdot \left(1 + \frac{N}{2} - \frac{1}{N+1} \right) \end{aligned} \quad (1.1)$$

1.3 Алгоритм поиска в упорядоченном словаре двоичным поиском

Бинарный поиск базируется на том, что словарь изначально отсортирован, что позволяет сравнивать ключ с средним элементом, и, если, он меньше, то продолжать искать в левой части, таким же методом, иначе в правой.

Таким образом, при двоичном поиске [3] обход можно представить деревом, поэтому трудоёмкость в худшем случае составит $k_0 + \log_2 N$ (в худшем случае нужно спуститься по двоичному дереву от корня до листа).

Лучшим случаем будет случай, если искомый ключ окажется средним элементом, тогда трудоемкость будет равна $k_0 + \log_2 1$.

Скорость роста функции $\log_2 N$ меньше, чем скорость роста линейной функции, полученной для полного перебора.

1.4 Вывод

В данном разделе был рассмотрен абстрактный тип данных словарь и возможные реализации алгоритмов поиска в нём.

Программа будет получать на вход ключ, по которому необходимо провести поиск в словаре. При неверном вводе ключа (пустая строка) будет выведено сообщение об ошибке.

Реализуемое ПО дает возможность получить лог программы для двух алгоритмов (в лог файле содержится количество сравнения при поиске каждого ключа). Также имеется возможность построения графиков зависимости времени поиска каждого ключа.

2 Конструкторская часть

В этом разделе будут представлено описание используемых типов данных, а также схемы алгоритмов поиска в словаре.

2.1 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие типы данных:

- словарь — встроенный тип `dict` [4] в Python[5] будет использован в созданном классе `Dictionary`;
- массив ключей — встроенный тип `list` [6] в Python[5];
- длина массива/словаря — целое число `int`.

2.2 Структура разрабатываемого ПО

В данном ПО будет реализован метод структурного программирования, при этом также будет реализован класс `Dictionary` для работы со словарем.

Взаимодействие с пользователем будет через консоль, будет дана возможность ввода ключа для поиска значений в словаре.

2.3 Схемы алгоритмов

На рисунке 2.1 представлена схема алгоритма поиска в словаре полным перебором.

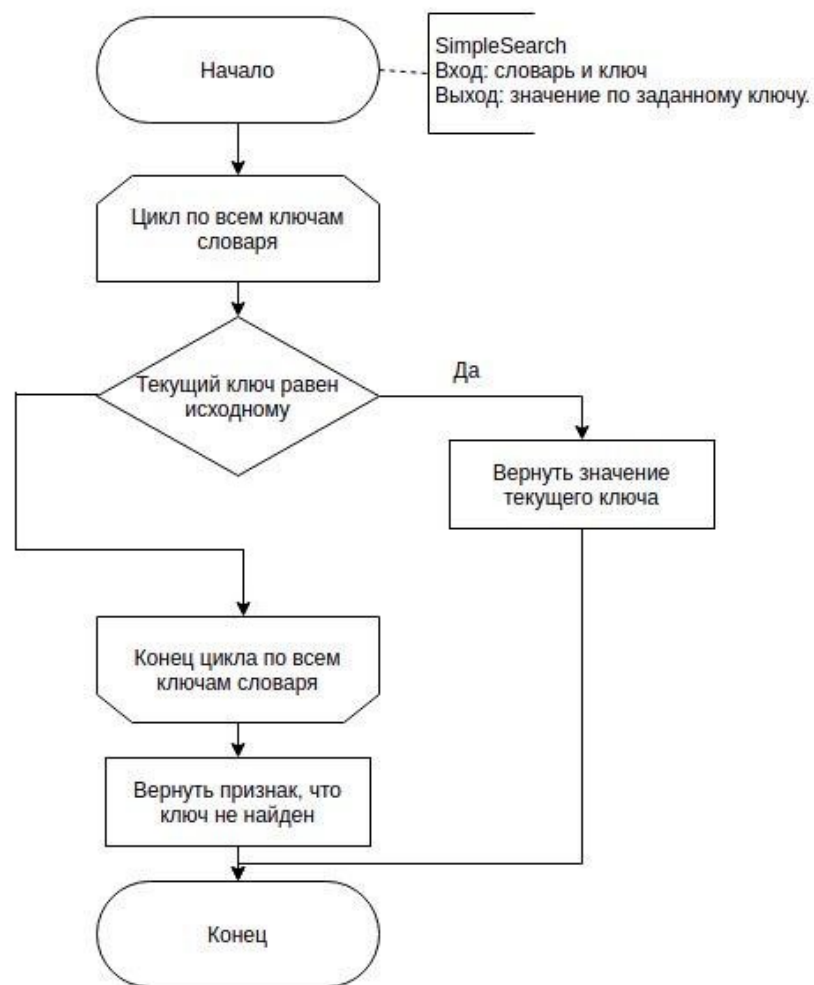


Рисунок 2.1 – Схема алгоритма поиска в словаре полным перебором

На рисунке 2.2 представлена схема алгоритма бинарного поиска в словаре.

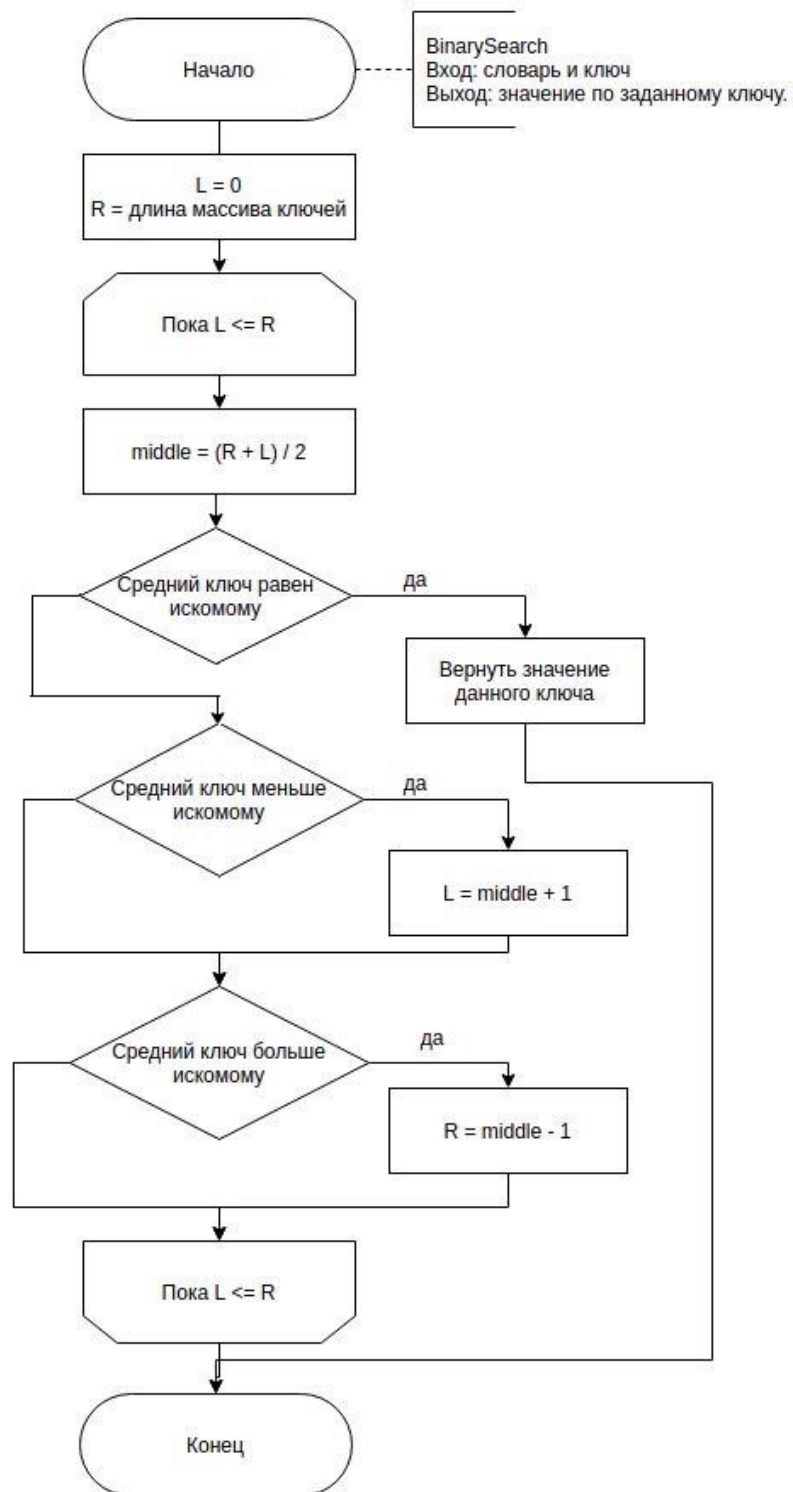


Рисунок 2.2 – Схема алгоритма бинарного поиска в словаре

2.4 Классы эквивалентности при тестировании

Для тестирования выделены классы эквивалентности, представленные ниже.

1. Некорректный ввод ключа – пустая строка.
2. Корректный ввод, но ключа нет в словаре – вывод будет -1, как знак того, что такого ключа нет словаре.
3. Корректный ввод и ключ есть в словаре – вывод верного значения.

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 приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Требования к ПО

К программе предъявляется ряд требований:

- на вход подаётся ключ, значение которого необходимо найти;
- на выходе — два результата поиска в словаре (для каждого алгоритма выводится отдельный результат).

3.2 Средства реализации

В качестве языка программирования (ЯП) для реализации данной лабораторной работы был выбран ЯП Python [5]. Данный язык достаточно удобен и гибок в использовании.

Время реализации работы алгоритмов было замерено с помощью функции `process_time()` из библиотеки `time` [7].

3.3 Реализация алгоритмов

В листинге 3.1 представлена реализация алгоритма поиска в словаре полным перебором.

Листинг 3.1 – Реализация алгоритма поиска полным перебором

```
1 def BruteForceSearch(self, key):
2     k = 0
3     kk = list(self.data.keys())
4     for elem in self.data:
5         k += 1
6         if key == elem:
7             // Writing to the log file
8             self.f.write(f"{kk.index(key)},{key},{k}\n")
9             return self.data[elem]
10    return -1
```

В листинге 3.2 представлена реализация алгоритма бинарного поиска в словаре.

Листинг 3.2 – Реализация алгоритма бинарного поиска

```
1 def BinarySearch(self, key, list_keys):
2     l, r = 0, len(list_keys) - 1
3     k = 0
4     kk = list(self.data.keys())
5     while l <= r:
6         middle = (r + l)
7         elem = list_keys[middle]
8         k += 1
9         if elem == key:
10            // Writing to the log file
11            self.f1.write(f"{kk.index(key)},{key},{k}\n")
12            return self.data[elem]
13        elif elem < key:
14            l = middle + 1
15        else:
16            r = middle - 1
17    return -1
```

3.4 Функциональное тестирование

В данном разделе будет приведена таблица с тестами.

Таблица 3.1 – Таблица тестов

Входные данные	Пояснение	Результат
Moscow	Средний элемент	Ответ верный
Tokyo	Первый элемент	Ответ верный
Belgium	Последний элемент	Ответ верный
123	Несуществующий элемент	Ответ верный (-1)
Mosccow	Несуществующий элемент	Ответ верный (-1)

Все тесты пройдены успешно для всех реализаций алгоритмов.

3.5 Вывод

В данном разделе были представлены листинги рассматриваемых алгоритмов поиска в словаре, приведена информация о средствах реализации, сведения о модулях программы и было проведено функциональное тестирование.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программа, а также проведены замеры процессорного времени и количества сравнений.

4.1 Технические характеристики

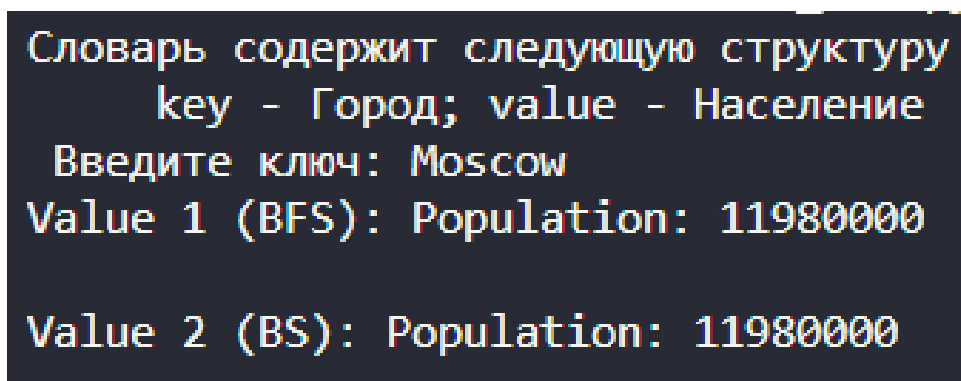
Технические характеристики устройства, на котором выполнялось тестирование:

- операционная система Window 10 Home Single Language;
- память 8 Гб;
- процессор 11th Gen Intel(R) Core(TM) i5-1135G7 2.42 ГГц, 4 ядра.

Во время замера устройство было подключено к сети электропитания, нагружено приложениями окружения и самой системой замера.

4.2 Демонстрация работы программы

На рисунках 4.1 и 4.2 представлены результаты работы программы.



```
Словарь содержит следующую структуру
      key - Город; value - Население
Введите ключ: Moscow
Value 1 (BFS): Population: 11980000
Value 2 (BS): Population: 11980000
```

Рисунок 4.1 – Пример 1 работы программы

```
Словарь содержит следующую структуру
    key - Город; value - Население
Введите ключ: 1
Value 1 (BFS): -1
Value 2 (BS): -1
```

Рисунок 4.2 – Пример 2 работы программы

4.3 Время выполнения реализаций алгоритмов поиска ключа

Как было сказано выше, для замера времени выполнения части кода используется функция `process_time()` из библиотеки `time` [7].

На рисунке 4.3 представлен график зависимости времени поиска от индекса ключа словаря, для построения которого использовались данные о времени поиска каждого элемента в словаре. Индекс ключа указан на горизонтальной оси.

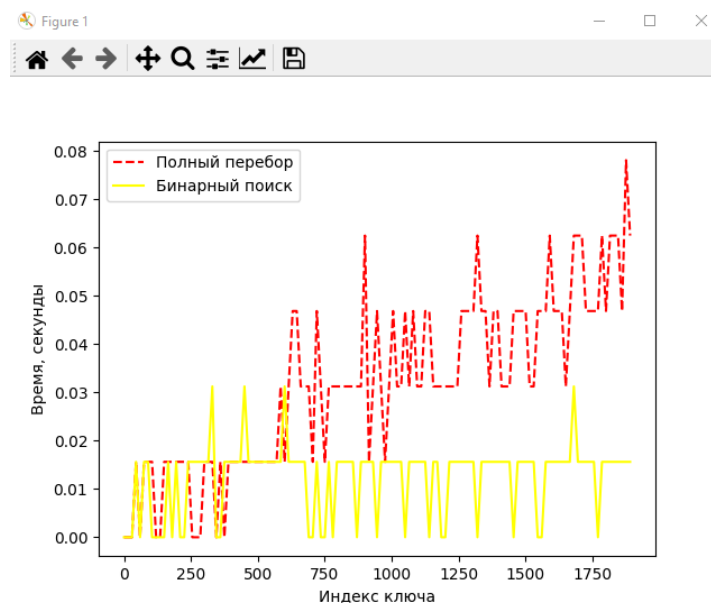


Рисунок 4.3 – Время работы алгоритмов для поиска каждого ключа словаря

4.4 Количество сравнений

В ходе эксперимента было подсчитано количество сравнений, которые понадобились, чтобы найти каждый ключ в словаре, и на основе полученных данных составлены гистограммы.

Гистограммы для алгоритма поиска в словаре полным перебором представлены на рисунке 4.4.

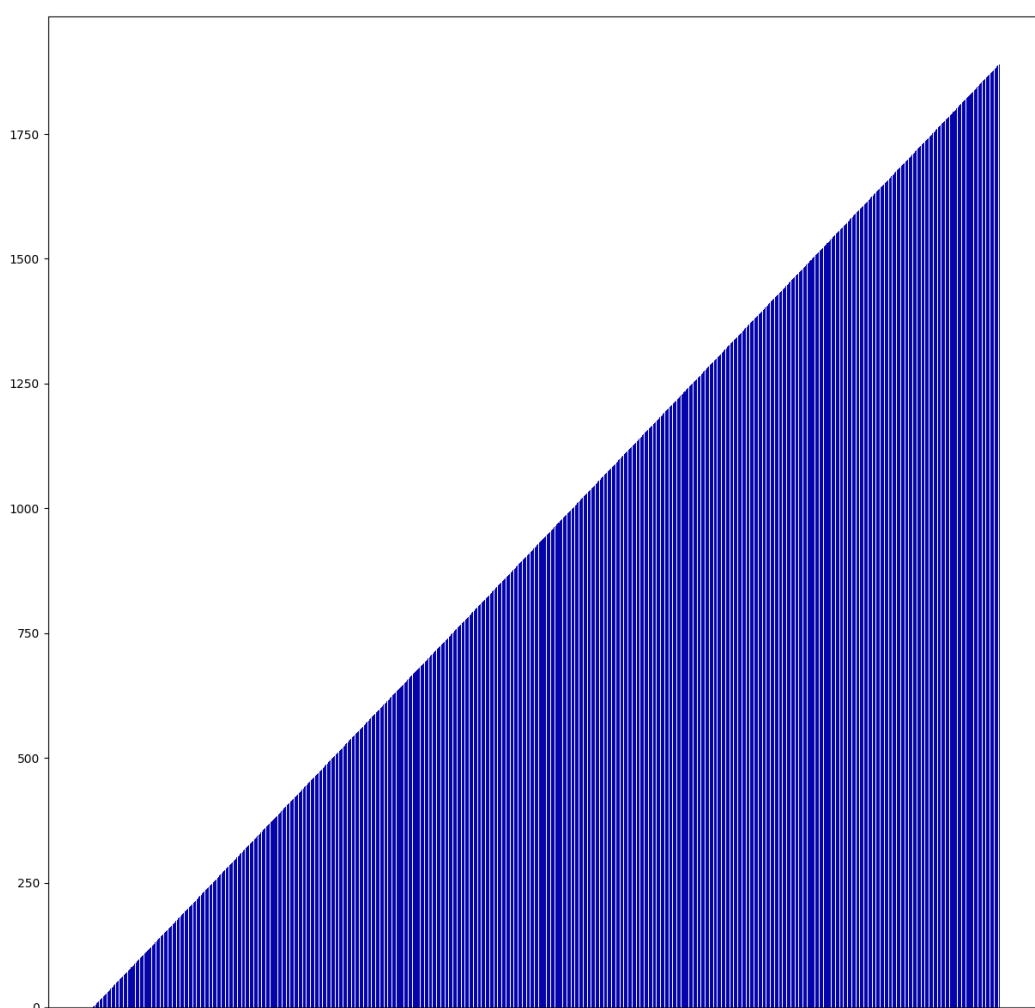


Рисунок 4.4 – Количество сравнений поиска с помощью полного перебора

Гистограммы для алгоритма бинарного поиска в словаре представлены на рисунках 4.5 и 4.6.

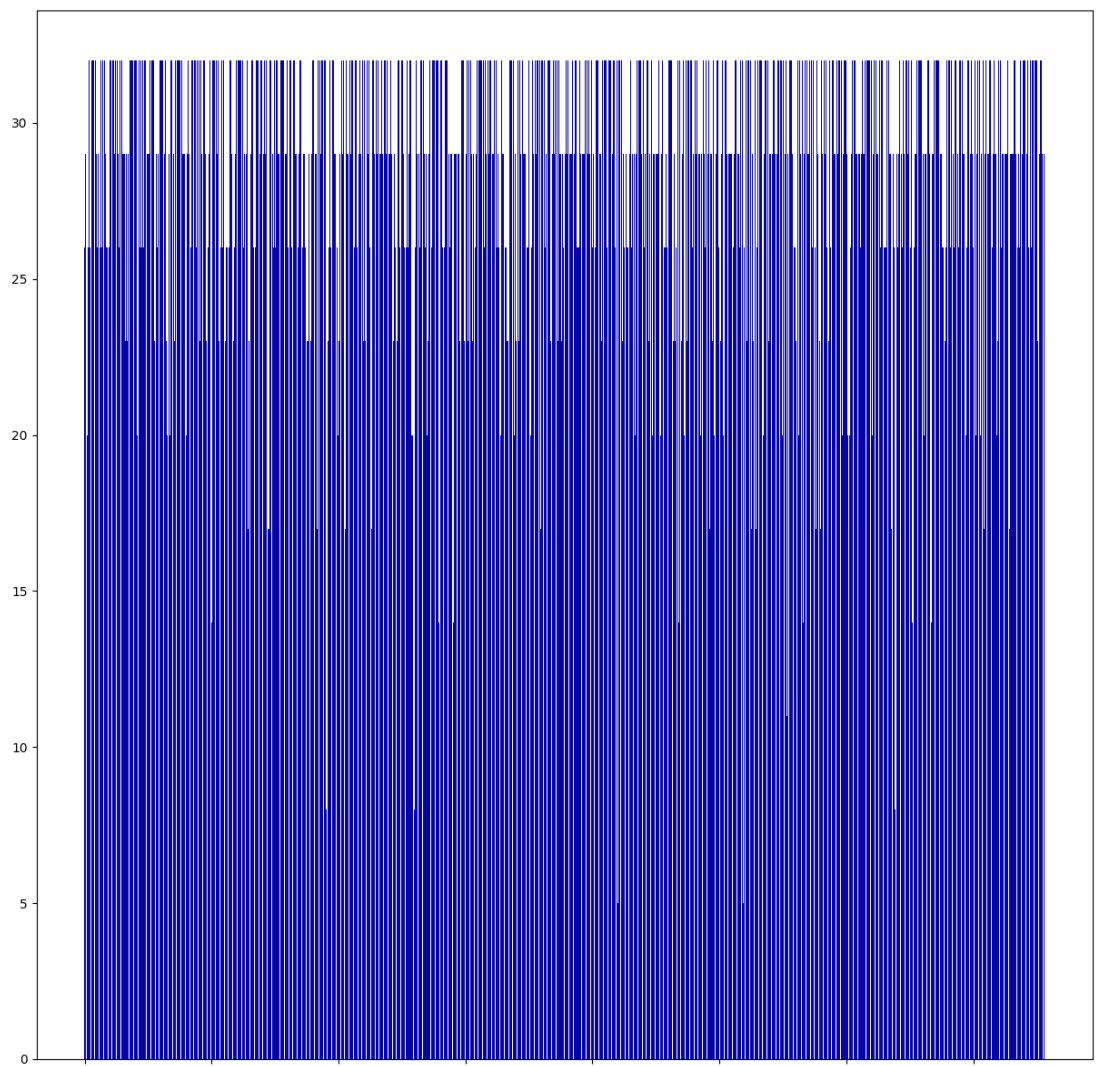


Рисунок 4.5 – Количество сравнений бинарного поиска, сортировка по расположению

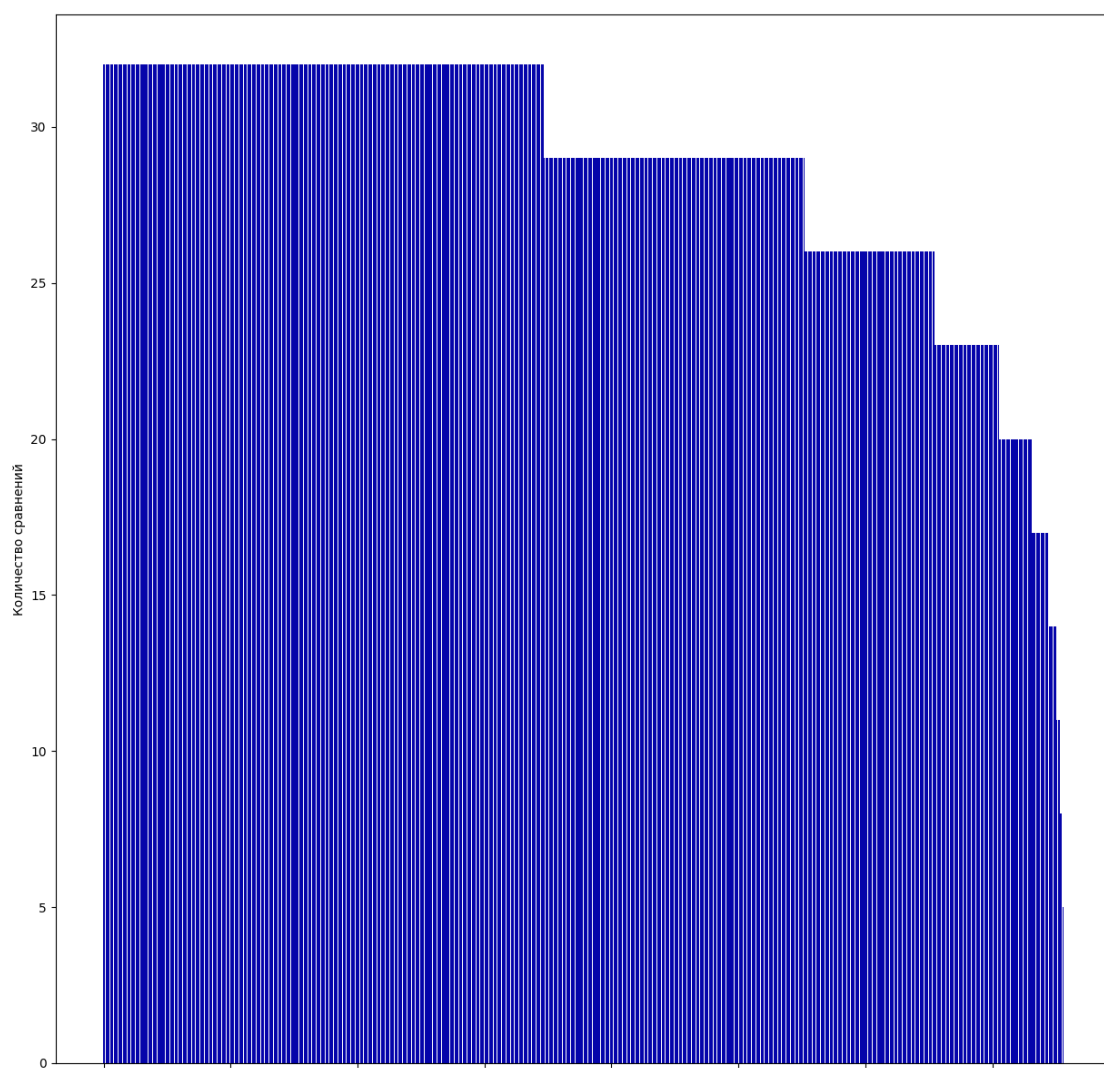


Рисунок 4.6 – Количество сравнений бинарного поиска, сортировка по количеству сравнений

4.5 Вывод

В результате эксперимента и полученных графиков и гистограмм, приведенных выше, видно, что алгоритм полного перебора медленнее, чем алгоритм бинарного поиска. Время в нём растет линейно и увеличивается с увеличением индекса элемента словаря. Также растет количество сравнений, что напрямую зависит от времени работы программы.

Алгоритм бинарного поиска требует дополнительных расходов времени на подготовку данных к работе с алгоритмом, но эти расходы можно не учитывать на этапе поиска (это предварительная подготовка данных).

Заключение

В результате эксперимента было определено, что алгоритм полного перебора работает медленнее. Но алгоритм бинарного поиска требует дополнительного времени на первичную обработку данных (сортировка).

В рамках выполнения работы были выполнены все задачи:

- Изучены два алгоритма поиска в словаре.
- Приведены схемы алгоритмов поиска в словаре.
- Описаны структуру разрабатываемого программного обеспечения.
- Применены изученные основы для реализации поиска значений в словаре по ключу.
- Проведено функциональное тестирование разработанного алгоритма.
- Получены замеры количества сравнений для каждого ключа (для всех двух алгоритмов).
- Проведен сравнительный анализ по времени для реализованных алгоритмов.
- Подготовлен отчет по лабораторной работе.

Поставленная цель достигнута.

Литература

1. National Institute of Standards and Technology [Электронный ресурс]. Режим доступа: <https://xlinux.nist.gov/dads/HTML/assocarray.html> (дата обращения 13.12.2020).
2. Н. Нильсон. Искусственный интеллект. Методы поиска решений. М.: Мир, 1973. с. 273.
3. Коршунов Ю. М. Коршунов Ю. М. Математические основы кибернетики // Энергоатомиздат. 1972.
4. dict Python [Электронный ресурс]. Режим доступа: <https://docs.python.org/3/library/2to3.html?highlight=dict#to3fixer-dict> (дата обращения: 04.09.2021).
5. Welcome to Python [Электронный ресурс]. Режим доступа: <https://www.python.org> (дата обращения: 04.09.2021).
6. list Python [Электронный ресурс]. Режим доступа: <https://docs.python.org/3/library/pdb.html?highlight=list#pdbcommand-list> (дата обращения: 04.09.2021).
7. time — Time access and conversions [Электронный ресурс]. Режим доступа: <https://docs.python.org/3/library/time.html#functions> (дата обращения: 04.09.2021).