



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
(ИУ7)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.04 Программная инженерия

Отчет по лабораторной работе №7 по дисциплине анализ алгоритмов

Тема: Поиск в словаре
Студент: Серова М.Н.
Группа: ИУ7-55Б
Оценка (баллы): _____
Преподаватель: Волкова Л.Л.

Москва, 2021

Оглавление

Введение	2
1 Аналитическая часть	3
1.1 Поиск полным перебором	3
1.2 Двоичный поиск	3
1.3 Сегментация словаря	4
1.4 Выводы	4
2 Конструкторская часть	5
2.1 Схемы алгоритмов	5
2.2 Структура ПО	7
2.3 Классы эквивалентности	7
2.4 Выводы	8
3 Технологическая часть	9
3.1 Средства реализации	9
3.2 Реализация алгоритмов	9
3.3 Трудоемкость	11
3.3.1 Трудоемкость полного перебора	11
3.3.2 Трудоемкость бинарного поиска	11
3.3.3 Трудоемкость поиска в словаре, разбитом на сегменты	11
3.4 Тестирование	11
3.5 Выводы технологической части	12
4 Экспериментальная часть	13
4.1 Пример работы программы	13
4.2 Технические характеристики	13
4.3 Сравнительный анализ алгоритмов по количеству сравнений	14
4.4 Выводы	16
Заключение	17
Список литературы	18

Введение

Словарь – структура данных, построенная на основе пар значений. Первое значение – ключ идентификатора, второе значение – сам элемент. Данная структура данных имеет широкое применение, например, при составлении справочников, языковых словарей, энциклопедий. Так как количество значений в словаре может быть очень большим, важно организовывать поиск данных по ключу максимально эффективным образом.

Цель – изучение алгоритмов поиска в словаре. Для достижения поставленной цели необходимо решить следующие задачи:

1. изучить способы поиска по ключу в словаре;
2. получить практические навыки реализации алгоритмов поиска полным перебором, бинарным поиском и поиском в словаре, разбитом на сегменты;
3. рассчитать теоретическую трудоемкость разработанных алгоритмов;
4. провести сравнительный анализ алгоритмов по количеству сравнений для каждого ключа;
5. описать полученные результаты в отчете по лабораторной работе, выполненном как расчётно-пояснительная записка к ней.

1 Аналитическая часть

Словарь – это ассоциативный массив, позволяющий хранить пары вида «ключ-данные». Зачастую полагается, что заданному ключу соответствует единственное значение данных. Данная структура поддерживает следующие операции:

- insert - добавление пары ключ-значение в словарь;
- find - поиск значения по заданному ключу;
- remove - удаление пары ключ-значения из словаря по заданному ключу.

В данной лабораторной работе словарь представляет собой Рыбную Энциклопедию, осуществляющую доступ к информации о семье, к которой принадлежит рыба, и опасности, которую эта рыба представляет для человека, по её имени. Словарь хранит 34000 значений.

1.1 Поиск полным перебором

Полный перебор – один из подходов реализации операции поиска. В этом случае поиск по ключу осуществляется последовательным сравнением существующих ключей с заданным. При данном подходе возможно существование $(N+1)$ ситуации при поиске: N возможных случаев расположения ключа в массиве/словаре и его отсутствие.

Данный подход не налагает никаких требований на существующую структуру, хранящие ключи, и не накладывает никаких ограничений на осуществление остальных операций.

1.2 Двоичный поиск

В случае, если есть возможность отсортировать ключи словаря, может быть использован двоичный поиск. При двоичном поиске массив делится на две половины и принадлежность элемента к одной из половин устанавливается единственным сравнением.

Данный процесс может быть описан следующим образом:

1. определение значения элемента в середине структуры данных, сравнение полученного значения с ключом;
2. осуществление поиска в первой половине, если ключ меньше значения середины, иначе – во второй;
3. в выбранной половине вновь определяется значение серединного элемента, проводится сравнение с ключом;
4. процесс продолжается до тех пор, пока не будет найден элемент со значением ключа или интервал для поиска не станет пустым.

И, хотя двоичный поиск значительно ускоряет скорость нахождения, его использование накладывает некоторые условия при модификации словаря/массива, что приводит к увеличению временных затрат на добавление или удаление ключа.

1.3 Сегментация словаря

Основной идеей этого способа является "разнесение" ключей словаря по сегментам, например, определяемых первой буквой слова. В этом случае задача поиска разбивается на два этапа: выбор сегмента словаря и поиск в самом сегменте.

В случае, если, сегменты отсортированы, может осуществляться двоичный поиск внутри этих сегментов. + доп техника

1.4 Выводы

Были рассмотрены способы поиска ключа в словаре. Входные данные – ключ, по которому происходит поиск, выходные – значение, которое привязано к этому ключу.

2 Конструкторская часть

Данный раздел содержит схемы реализуемых алгоритмов.

2.1 Схемы алгоритмов

Схема алгоритма поиска полным перебором представлена на рисунке 2.1.

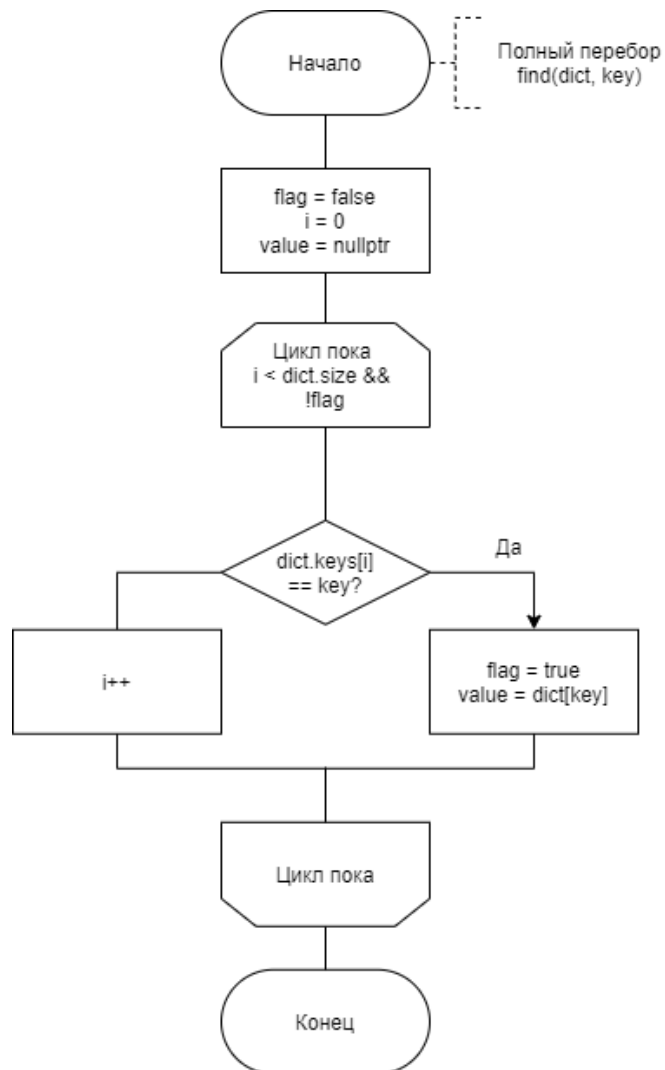


Рис. 2.1: Полный перебор

Схема алгоритма бинарным поиском представлена на рисунке 2.2.

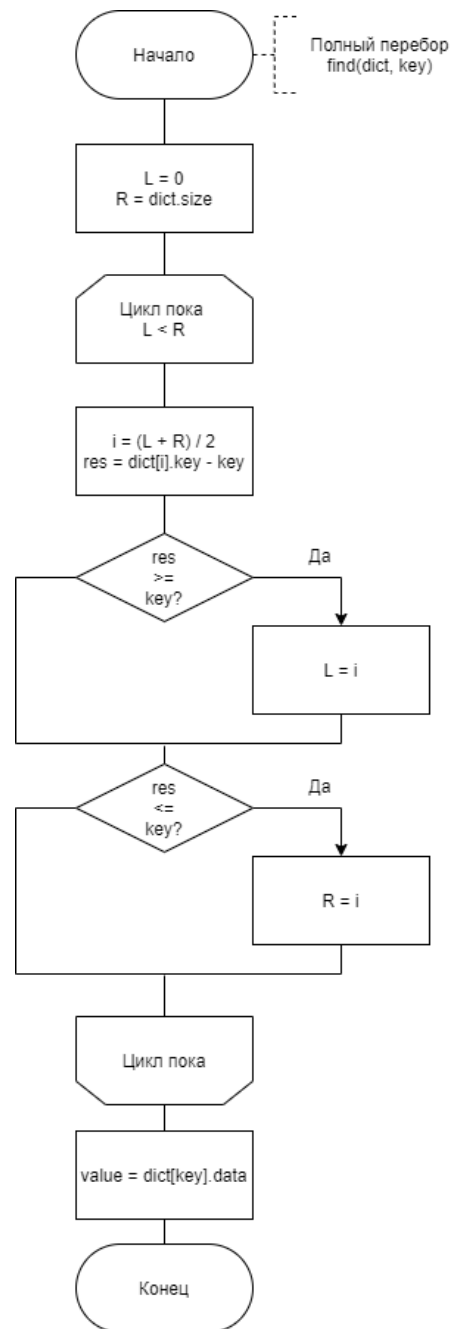


Рис. 2.2: Бинарный поиск

Схема алгоритма поиска в словаре при его сегментировании представлена на рисунке 2.3.

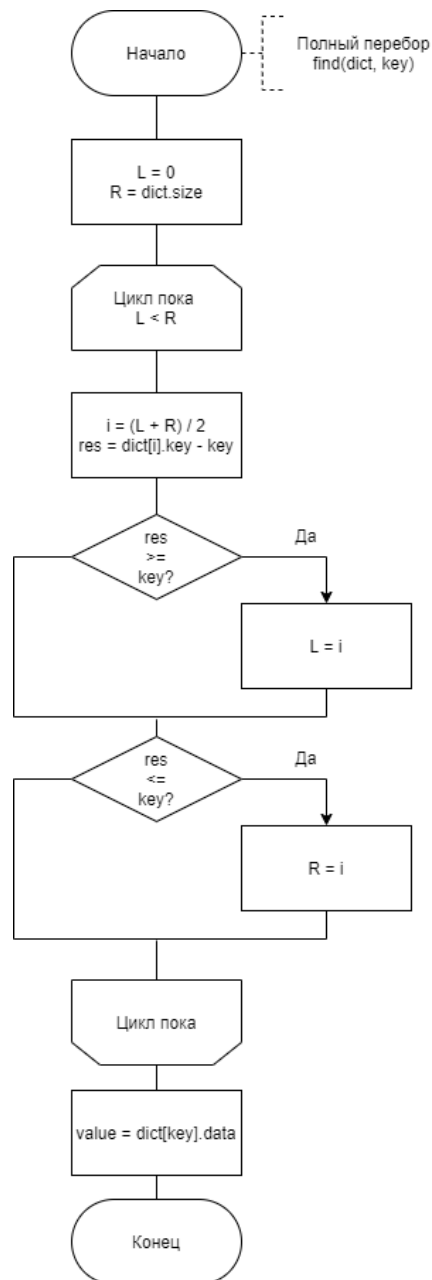


Рис. 2.3: Поиск в сегментах

2.2 Структура ПО

В качестве аргумента командной строки программе передаётся файл, в котором содержится информация для словаря, и строки, являющиеся ключами, по которым осуществляется поиск.

2.3 Классы эквивалентности

Для осуществления функционального тестирования ПО были выделены следующие классы эквивалентности:

- поиск первого ключа;

- поиск последнего ключа;
- поиск несуществующего ключа;
- поиск четного ключа;
- поиск нечетного ключа.

2.4 Выводы

На основе теоретических данных, приведенных в аналитическом разделе, были составлены схемы алгоритмов для их дальнейшей реализации в технологической части.

3 Технологическая часть

Данный раздел содержит обоснование выбора языка и среды разработки, реализацию алгоритмов.

3.1 Средства реализации

Для реализации программы был выбран язык программирования Python [1]. Такой выбор обусловлен следующими причинами:

- имеется большой опыт разработки;
- имеет большое количество расширений и библиотек, в том числе библиотеку для работы с потоками, измерения времени, построения графиков;
- обладает информативной документацией;

3.2 Реализация алгоритмов

В листингах 3.1 - 3.3 представлены реализации рассматриваемых алгоритмов.

Листинг 3.1: Реализация алгоритма поиска полным перебором

```
1 def search_simple(key, animals_dict):  
2     cnt = 0  
3     for dict_key in animals_dict.keys():  
4         cnt += 1  
5         if key == dict_key:  
6             return animals_dict[key], cnt  
7     return None, 0
```

Листинг 3.2: Реализация алгоритма бинарного поиска

```
1 def bin_search(key, animal_dict):
2     keys = list(animal_dict.keys())
3     cnt = 0
4     i = 0
5     j = len(animal_dict) - 1
6     m = int(j / 2)
7
8     while keys[m] != key and i < j:
9         cnt += 1
10        if key > keys[m]:
11            i = m + 1
12        else:
13            j = m - 1
14        m = int((i + j) / 2)
15    cnt += 1
16    if keys[m] != key:
17        return None, cnt
18    else:
19        return animal_dict[keys[m]], cnt
```

Листинг 3.3: Реализация алгоритма поиска по сегментам

```
1 def combined_search(key, seg_dict):
2     cur_dict = {}
3     a = key[0]
4     cnt = 0
5     if a < 'A' or a > 'Z':
6         return None, 0
7     else:
8         for i in seg_dict.keys():
9             cnt += 1
10            if i == key[0]:
11                cur_dict = seg_dict[i]
12                break
13        if len(cur_dict) == 0:
14            return None, 0
15        r, c = bin_search(key, cur_dict)
16        return r, c+cnt
```

3.3 Трудоемкость

Так как функцией, оказывающей основную нагрузку на алгоритм является сравнение строк-ключей, теоретическая трудоемкость определяется как необходимое количество сравнений.

3.3.1 Трудоемкость полного перебора

Лучший случай для алгоритма поиска полным перебором наступает, если искомый ключ находится в первом элементе массива. В этом случае количество сравнений ключей определяется, как $f_{best} = 1$. В худшем случае, если ключ находится в последнем элементе массива или не содержится в нем, количество сравнений равняется $f_{worst} = N = 3350$. Медианная трудоемкость определяется, как $f_{median} = \frac{\sum_{i=1}^N (i) + N}{N} \approx \frac{1+N}{2N} \cdot N + N \approx \frac{N}{2} = 1675$

3.3.2 Трудоемкость бинарного поиска

Лучший случай для алгоритма бинарного поиска наступает, если искомый ключ находится точно в середине массива. В этом случае количество сравнений ключей определяется, как $f_{best} = 1$. В худшем случае, если ключ, например, отсутствует в массиве, количество сравнений равняется $f_{worst} = \log N + 1 = 13$. Медианная трудоемкость определяется, как $f_{median} = \frac{\sum_{i=1}^{\log N} (i \cdot 2^{i+1})}{N} = \frac{2^{\log N} \cdot \log N - 2^{\log N} + 1}{N} \approx \frac{N \cdot \log N - N}{N} \approx \log N = 12$

3.3.3 Трудоемкость поиска в словаре, разбитом на сегменты

Поиск в словаре, разбитом на сегменты, аналогично поиску полным перебором требует $f_{best} = 2$ сравнений в лучшем случае (элемент находится в первом сегменте в начале). Худший случай наступает для последнего сегмента, состоящего из наименьшего количества значений (5) и равняется, соответственно, $f_{worst} = \log 5 + 26 = 28$. Для сформированного словаря медианный случай требует в среднем $f_{median} = 14.4$ сравнений

3.4 Тестирование

Тестирование реализованных алгоритмов проходило по методу "черного ящика". В таблице 3.1 описаны проверяемые тестовые случаи. Столбцу "вид" соответствует название биологического вида, столбцу "Научное название" соответствующая информация. Фактические результаты тестов совпали с ожидаемыми.

Таблица 3.1: Таблица тестовых данных

Вид	Научное название
Aardwolf	Proteles cristatus
Zungaropsis multimaculatus	Zwackhiomyces dispersus (J. Lahm ex Körb.) Triebel & Grube
aedfgh	Ключ не найден
Ginseng	Panax pseudoginseng Wall.
Florida Sphagnum	Sphagnum macrophyllum Brid. var. floridanum Austin

3.5 Выводы технологической части

В данном разделе были реализованы и протестированы алгоритмы поиска в словаре по ключу.

Вывод

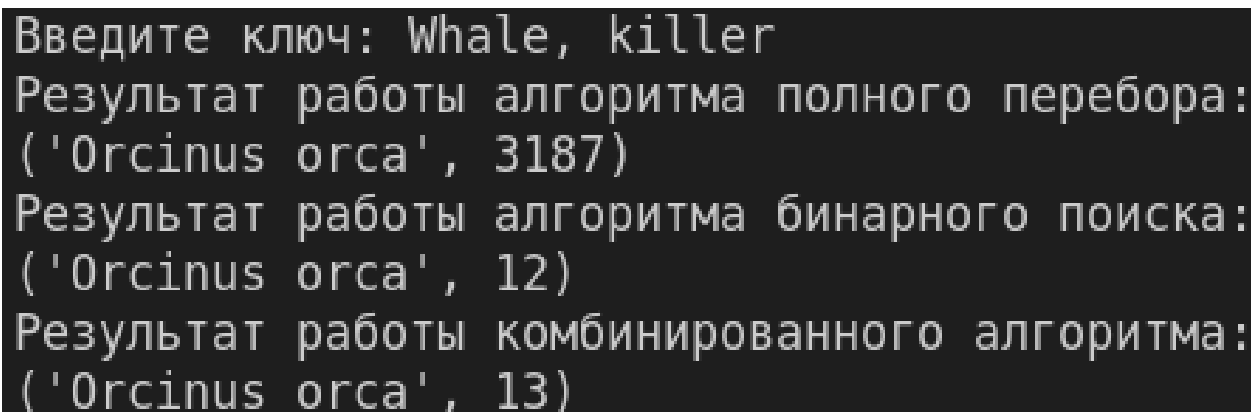
В этом разделе обоснован выбор языка программирования, описаны технические характеристики, приведены листинги кода и оценка трудоемкости реализованных алгоритмов, проведены тесты.

4 Экспериментальная часть

В данном разделе сравниваются реализованные алгоритмы, дается сравнительная оценка затрат на время.

4.1 Пример работы программы

Пример работы программы представлен на рисунке 4.1.



```
Введите ключ: Whale, killer
Результат работы алгоритма полного перебора:
('Orcinus orca', 3187)
Результат работы алгоритма бинарного поиска:
('Orcinus orca', 12)
Результат работы комбинированного алгоритма:
('Orcinus orca', 13)
```

Рис. 4.1: Пример работы программы

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование:

- операционная система: Ubuntu 20.01 Linux x86_64 [2];
- оперативная память: 8 Гб;
- процессор: AMD Ryzen5 4500U [3]:
 - количество физических ядер: 6;
 - количество логических ядер: 6.

В данном разделе проводится сравнительный анализ реализованных алгоритмов по затратам процессорного времени.

4.3 Сравнительный анализ алгоритмов по количеству сравнений

Гистограмма, визуализирующая необходимое количество сравнений для нахождения заданного ключа полным перебором представлена ниже, на рисунке 4.2.

Аналогичные рисунки представлены для алгоритма бинарного поиска (4.3), а также для поиска в сегментированном словаре (4.5).

На рисунках 4.4 и 4.6 изображено, какие ключи будут найдены в массиве ключей за указанное число сравнений для алгоритмов бинарного поиска и поиска в сегментированном словаре соответственно, для алгоритма полного перебора такая гистограмма совпадает с гистограммой, описанной выше.

На представленных диаграммах видно, что количество сравнений при поиске полным перебором прямо пропорционально номеру, под которым ключ находится в массиве. Количество сравнений при бинарном поиске не имеет какой-либо выраженной зависимости от номера ключа. Для сегментированного словаря количество сравнений зависит от принадлежности конкретному сегменту, поэтому общее количество сравнений выше, чем для алгоритма бинарного поиска, что обуславливается довольно малым размером исходного словаря (3349 ключей) и неравномерным распределением значений в сегментах.

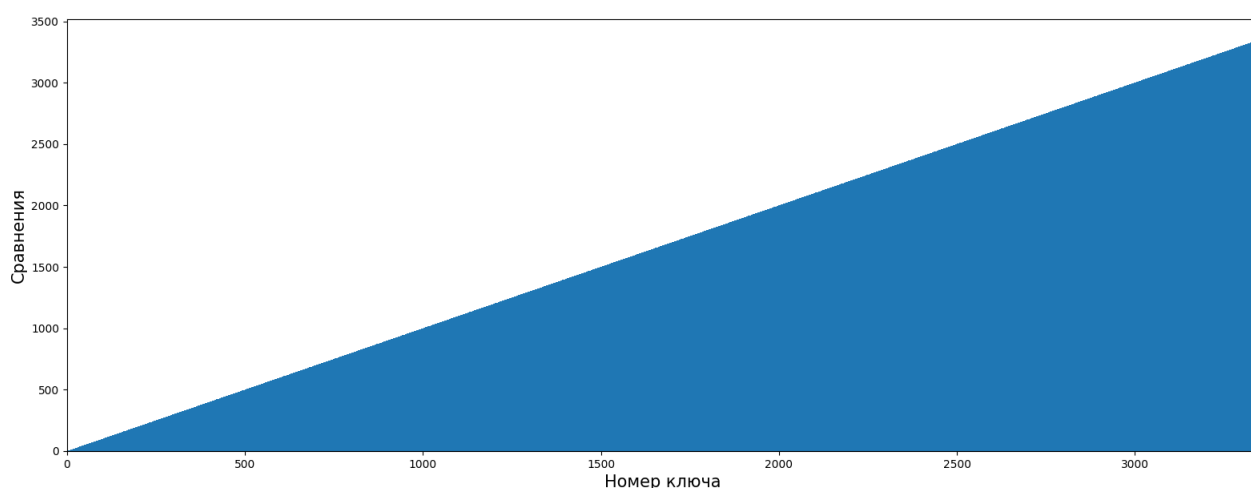


Рис. 4.2: Полный перебор

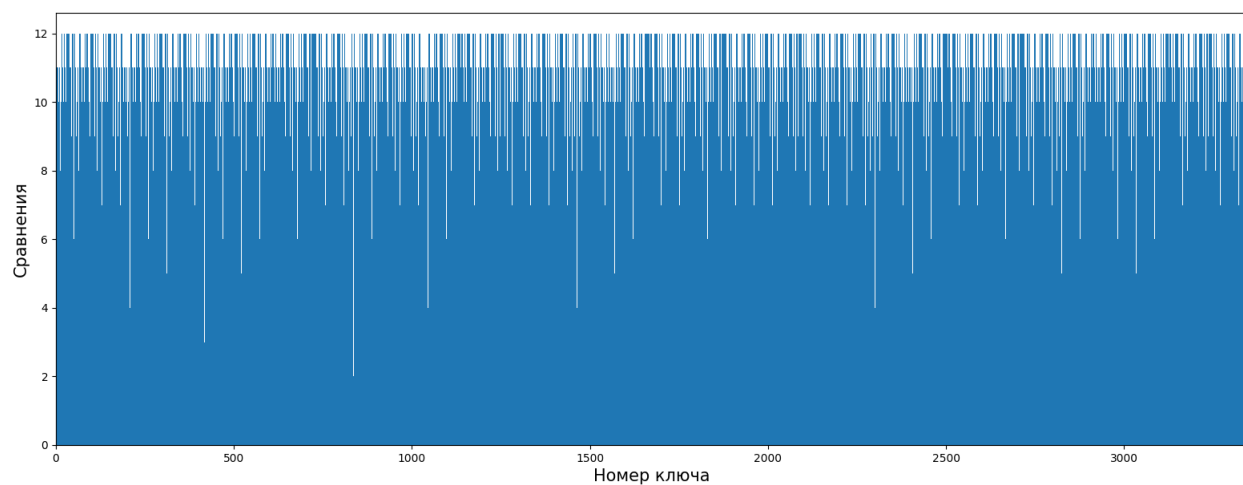


Рис. 4.3: Бинарный поиск (часть 1)

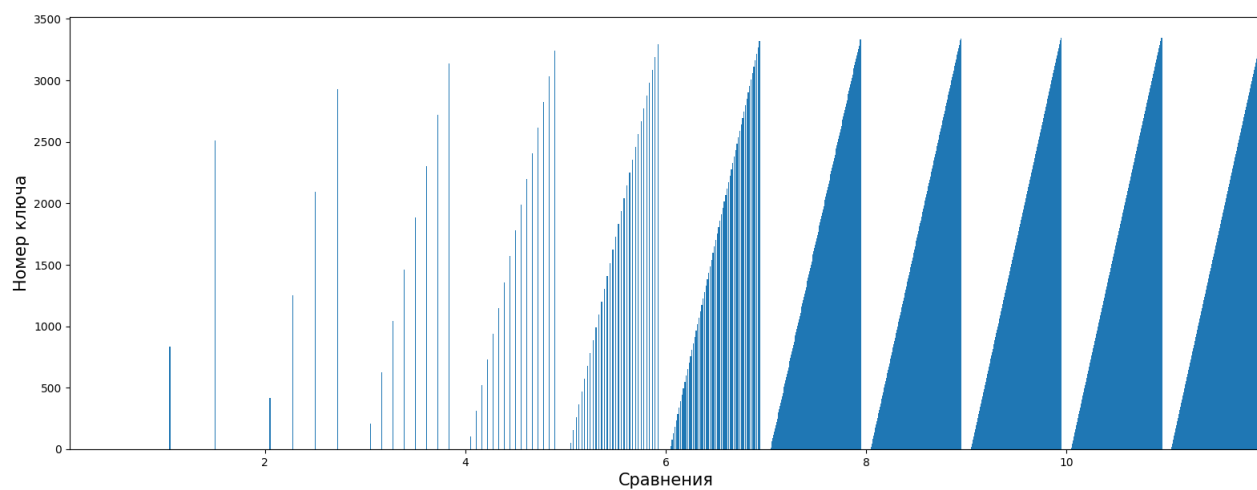


Рис. 4.4: Бинарный поиск (часть 2)

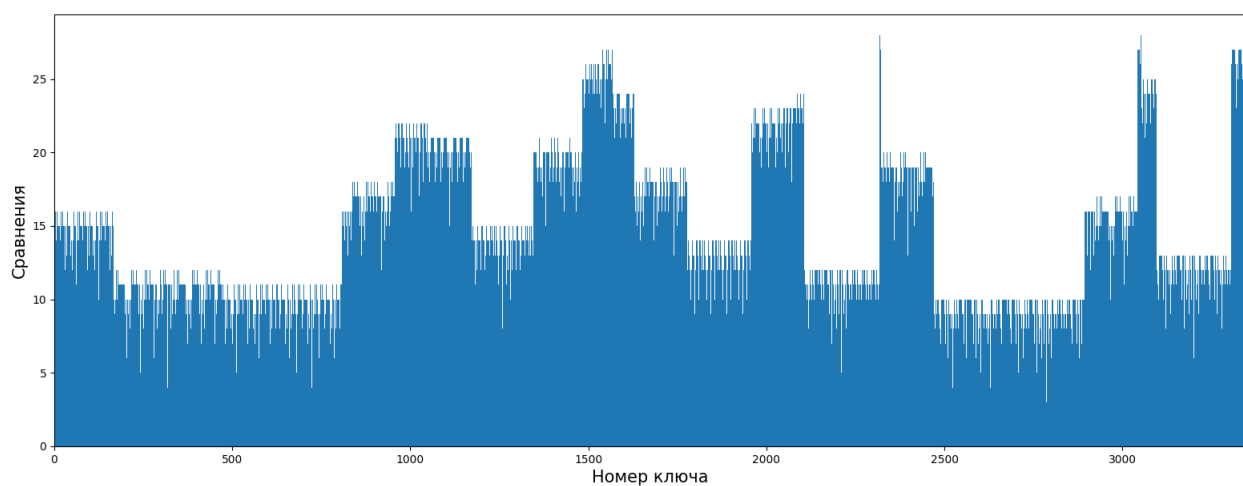


Рис. 4.5: Сегментированный словарь (часть 1)

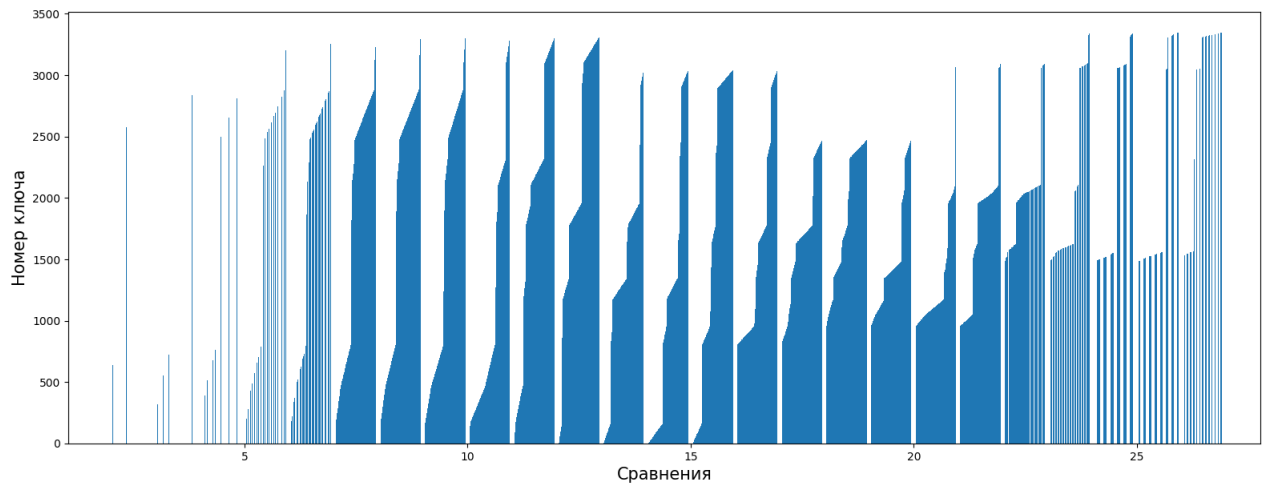


Рис. 4.6: Сегментированный словарь (часть 2)

4.4 Выводы

Бинарный алгоритм требует в среднем меньше сравнений, чем алгоритм полным перебором. Максимальное количество сравнений при поиске в сегментированном словаре больше аналогичного показателя при бинарном поиске. При одинаковом количестве сравнений, поиск в сегментированном словаре найдет ключ чаще, чем при бинарном поиске. Поиск в сегментированном словаре тем эффективней, чем равномерней распределены значения исходного массива по сегментам.

Заключение

В процессе выполнения работы изучены алгоритмы поиска в словаре.

Все поставленные задачи выполнены.

1. Изучены способы поиска по ключу в словаре.
2. Реализованы и протестированы алгоритмы поиска полным перебором, бинарным поиском и бинарным поиском в сегментированном словаре.
3. Определена теоретическая трудоемкость алгоритмов, выраженная в количестве необходимых сравнений.
4. Экспериментально установлено уменьшение максимального необходимого числа сравнений и среднего необходимого числа сравнений при бинарном поиске в случае, если словарь является сегментированным.
5. Определены шаблоны распределения ключей от количества проведенных сравнений при бинарном поиске по массиву ключей.

Список литературы

- [1] About Python [Электронный ресурс]. Режим доступа: <https://www.python.org/about/>. Дата обращения: 20.10.2021.
- [2] Ubuntu по-русски [Электронный ресурс]. Режим доступа: <https://ubuntu.ru/>. Дата обращения: 15.12.2021.
- [3] AMD Ryzen™ 5 4500U. Режим доступа: <https://www.amd.com/ru/products/apu/amd-ryzen-5-4500u>. Дата обращения: 15.12.2021.
- [4] Ульянов М.В. Ресурсно эффективные алгоритмы. Разработка и анализ. //Наука Физматлит - Москва, 2007. с. 387.