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 дисциплине "Анализ алгоритмов"

Тема Поиск в словаре

Студент Малышев И. А.

Группа ИУ7-51Б

Оценка (баллы) _____

Преподаватель: Волкова Л. Л.

Москва — 2021 г.

Содержание

Введение

Словарь – структура данных, построенная на основе пар значений: ключ-значение. Ключ нужен для идентификации элементов, а значение – это собственно сам хранимый элемент. Например, в телефонном справочнике номеру телефона соответствует фамилия абонента. Существует несколько основных различных реализаций словаря: массив, двоичные деревья поиска, хеш-таблицы. Каждая из этих реализаций имеет свои минусы и плюсы, например время поиска, вставки и удаления элементов.

Целью данной лабораторной работы является изучение, реализация и сравнение алгоритмов поиска в словаре.

В рамках выполнения работы необходимо решить следующие **задачи**:

- реализовать три алгоритма поиска в словаре: линейный, бинарный и поиск с частотным анализом;
- замерить количество сравнений при поиске и сравнить данные между алгоритмами;
- сделать выводы на основе проделанной работы.

1 Аналитическая часть

В данном разделе представленные теоретические сведения о рассматриваемых алгоритмах.

1.1 Полный перебор

Алгоритм полного перебора заключается в проходе по словарю, до того момента, пока не будет найден искомый ключ. В рассматриваемом алгоритме возможно $N + 1$ случаев расположения ключа: ключ является i-ым элементом словаря либо его нет в словаре в принципе.

Лучший случай (трудоемкость $O(1)$): ключ расположен в самом начале словаря и найден за одно сравнение). Худший случай (трудоемкость $O(N)$): ключ расположен в самом конце словаря либо ключ не находится в словаре.

1.2 Двоичный поиск

Данный алгоритм подходит только для заранее упорядоченного словаря.
Процесс двоичного поиска можно описать следующим образом:

- получить значение находящееся в середине словаря и сравнить его с ключом;
- в случае, если ключ меньше данного значения, продолжить поиск в младшей части словаря, в обратном случае – в старшей части словаря;
- на новом интервале снова получить значение из середины этого интервала и сравнить с ключом.
- поиск продолжать до тех пор, пока не будет найден искомый ключ, или интервал поиска не окажется пустым.

Обход словаря данным алгоритм можно представить в виде дерева, поэтому трудоемкость в худшем случае составит $\log_2 N$. Можно сделать вывод, что алгоритм двоичного поиска работает быстрее чем алгоритм полного перебора, но при этом требует предварительной обработки данных (сортировки)[1].

1.3 Частотный анализ

Данный алгоритм также требует предварительной обработки данных, а именно:

- упорядочить словарь;
- разбить словарь на сегменты (сегментация).

Словарь разбивается на сегменты по какому-либо признаку и сортируется по частоте. Например, если ключ является строкой, то можно сделать разбиение по первой букве в ключе. Если ключ является целым числом, можно провести разбиение по остатку от деления ключа на некоторое число K.

После выполнения разбиения, нужно определить к какому сегменту относится искомый ключ и провести на этом сегменте двоичный поиск.

Таким образом, время поиска в словаре увеличивается (особенно для самых часто встречаемых ключей), но, при этом, так же как и алгоритм двоичного поиска, частотный анализ требует предварительной обработки данных (сортировки и сегментации)[2].

1.4 Описание словаря

Ключом в рассматриваемом мною словаре является Имя персонажа из книги «Божественная комедия»[3], а значением – его описание. Оба значения являются строковыми. Так как значение ключа является строкой, деление на сегменты будет производится посредством получения первой буквы в ключе.

Вывод

В данном разделе были рассмотренны особенности алгоритмов поиска в словаре.

2 Конструкторская часть

В данном разделе представлены схемы рассматриваемых алгоритмов.

2.1 Разработка алгоритмов

На рисунках 2.1 - 2.3 приведены схема алгоритмов поиска в словаре.

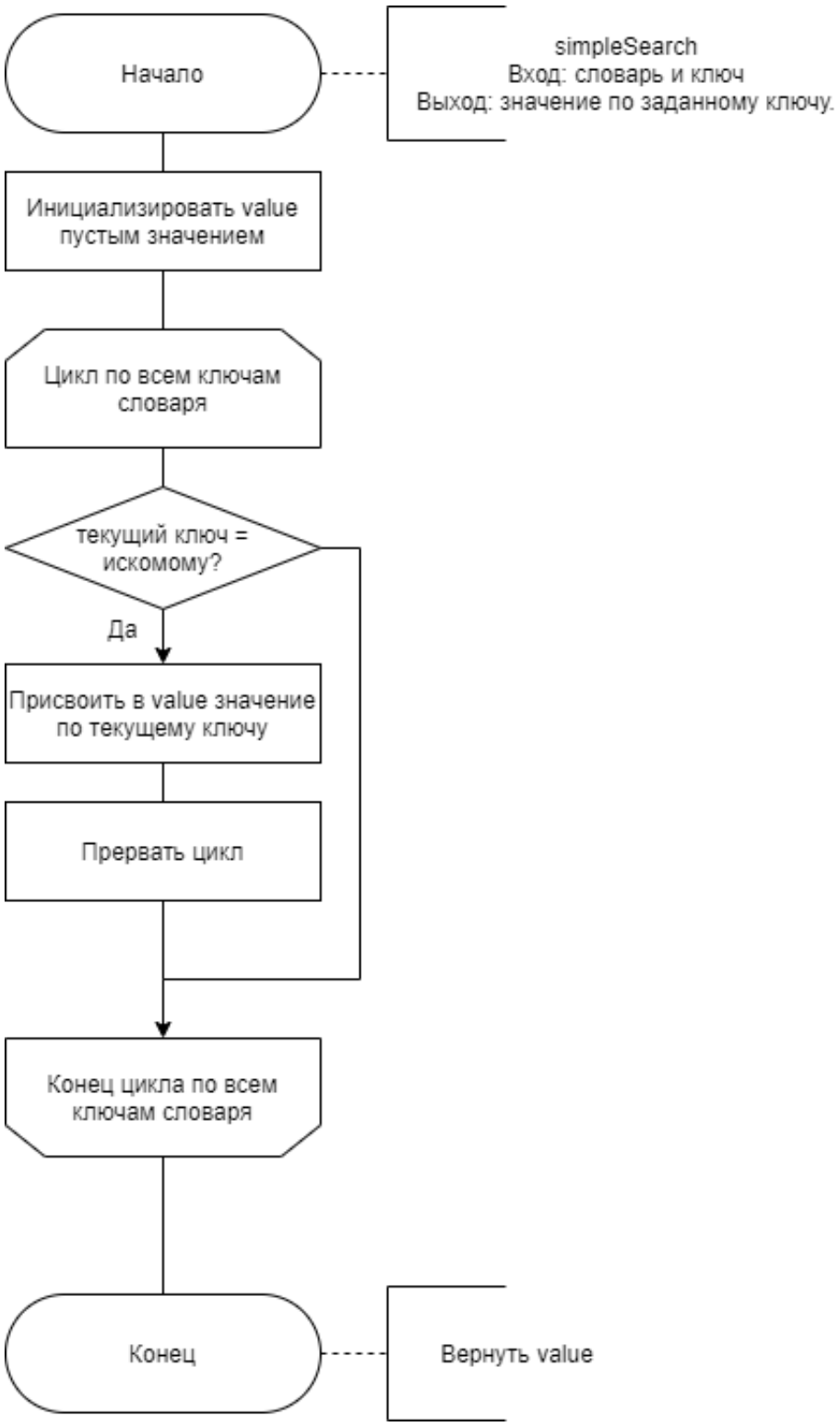

Рис. 2.1: Схема алгоритма полного перебора.

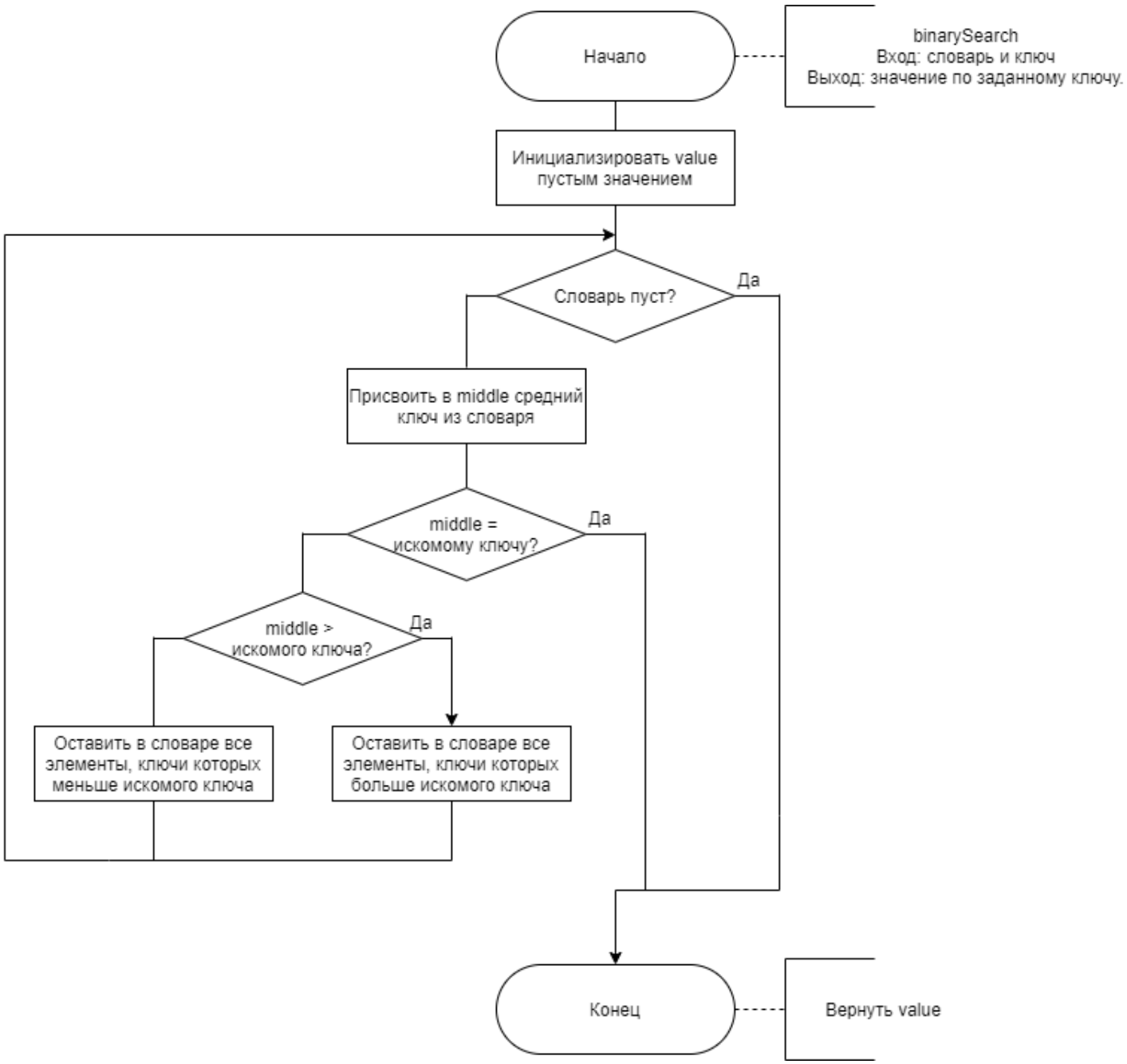

Рис. 2.2: Схема алгоритма двоичного поиска.

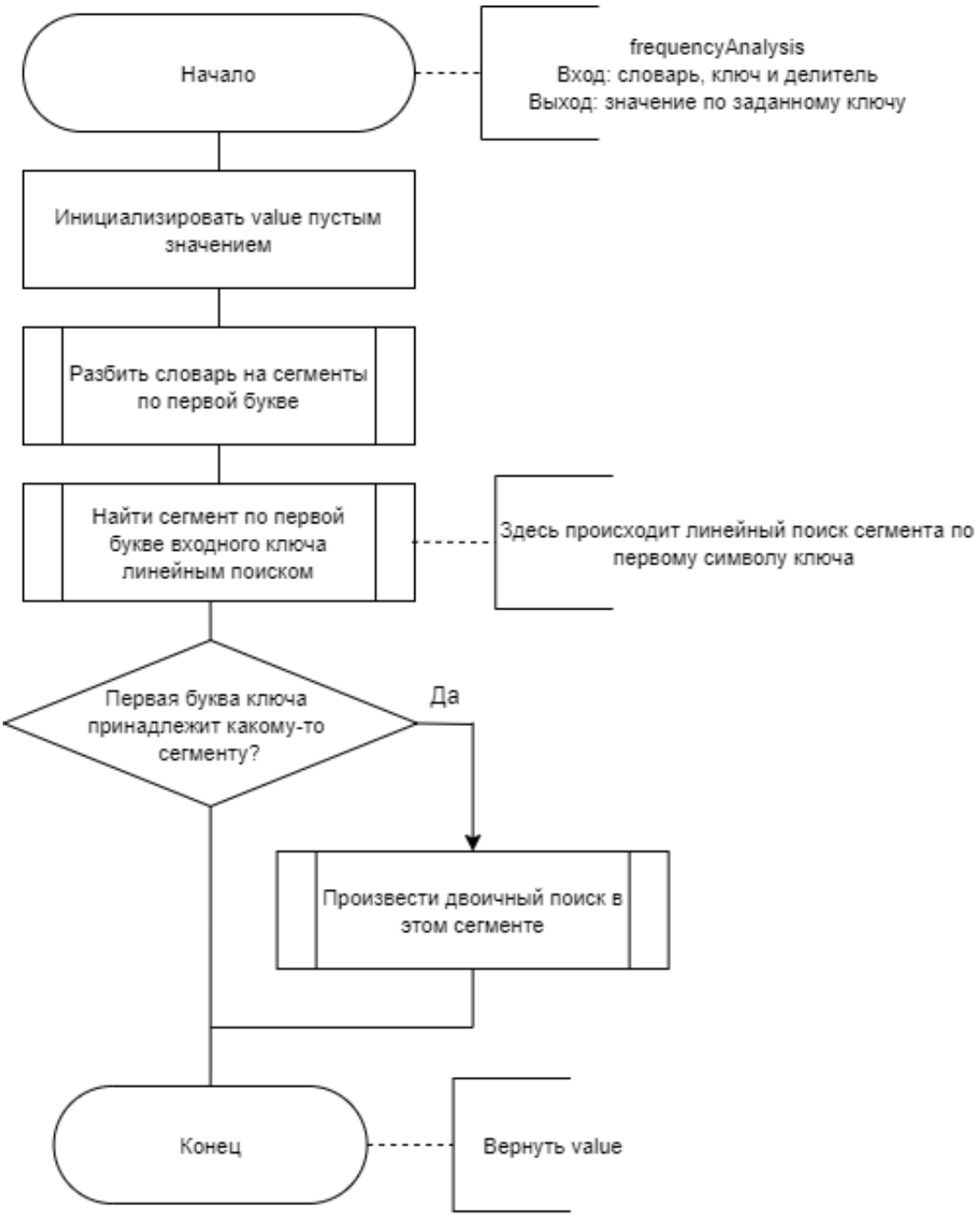

Рис. 2.3: Схема алгоритма поиска с использованием частотного анализа.

Вывод

На основе теоретических данных, полученных аз аналитического раздела, были построенны схемы алгоритмов поиска в словаре.

3 Технологическая часть

В данном разделе приведены средства реализации и листинги кода.

3.1 Средства реализации

В качестве языка программирования был выбран C# [4], а среды разработки – Visual Studio[5], т. к. я знаком с данным языком и имею представление о тестировании программ в данном языке.

3.2 Реализация алгоритмов

В листингах 3.1 - 3.3 представлены листинги алгоритмов поиска в словаре.

Листинг 3.1: Алгоритм полного перебора

```
1  public string LinearSearch(string key, out int cmp_cnt)
2  {
3    cmp_cnt = 0;
4    foreach (var d in dict)
5    {
6      cmp_cnt++;
7      if (d.Key == key)
8        return d.Value;
9    }
10
11   return null;
12 }
```

Листинг 3.2: Алгоритм двоичного поиска

```
public string BinarySearch(string key, int left, int right, out int
    cmp_cnt)
{
  int cmp, mid;
  cmp_cnt = 0;

  while (left <= right)
  {
    mid = left + (right - left) / 2;

    cmp_cnt++;
    if ((cmp = key.CompareTo(this[mid].Key)) > 0)
      left = mid + 1;
    else if (cmp < 0)
      right = mid - 1;
    else
      return dict[mid].Value;
  }

  return null;
}
```

Листинг 3.3: Алгоритм поиска с использованием частотного анализа

```
public void Clusterize()
{
  for (int i = 0; i < dict.Count; i++)
  {
    string key = dict[i].Key[0].ToString();
    if (clusters.ContainsKey(key))
      clusters[key] = new KeyValuePair<int,
         int>(Math.Min(clusters[key].Key, i),
         Math.Max(clusters[key].Value, i));
    else
      clusters.Add(key, new KeyValuePair<int, int>(i, i));
  }
}

public string ClusterSearch(string key, out int cmp_cnt)
{
  string cluster_key = key[0].ToString();
  return BinarySearch(key, clusters[cluster_key].Key,
     clusters[cluster_key].Value, out cmp_cnt);
}
```

3.3 Результаты тестирования

В таблице 3.1 приведены тестовые данные, где ПП – полный перебор, ДП – двоичный поиск, ЧА – частотный анализ. Все тесты были пройденны успешно.

Таблица 3.1: Таблица тестовых данных алгоритмов поиска в словаре.

Входные данные	Ожидаемый результат	ПП	ДП	ЧА
Ahithophel	See Absalom	See Absalom	See Absalom	See Absalom
Michael	Archangel	Archangel	Archangel	Archangel
Ulysses	See Odysseus	See Odysseus	See Odysseus	See Odysseus
erhgrgher	null	null	null	null

Вывод

В данном разделе были разработаны и протестированны алгоритмы поиска в словаре.

4 Исследовательская часть

В данном разделе приведен анализ характеристик разработанного ПО.

4.1 Результаты анализа

На рисунках 4.1 - 4.5 приведены результаты анализа алгоритмов поиска в словаре. Индекс ключа – положение ключа в словаре. Для случаев двоичного поиска и поиска с частотным анализом словари были приведены к требуемому виду.

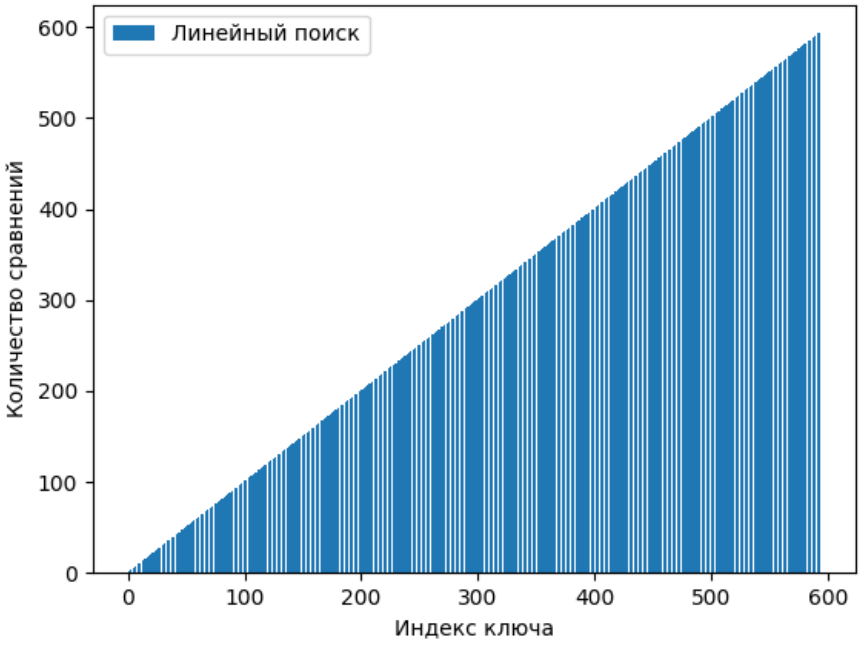

Рис. 4.1: Анализ количества сравнений для линейного поиска.

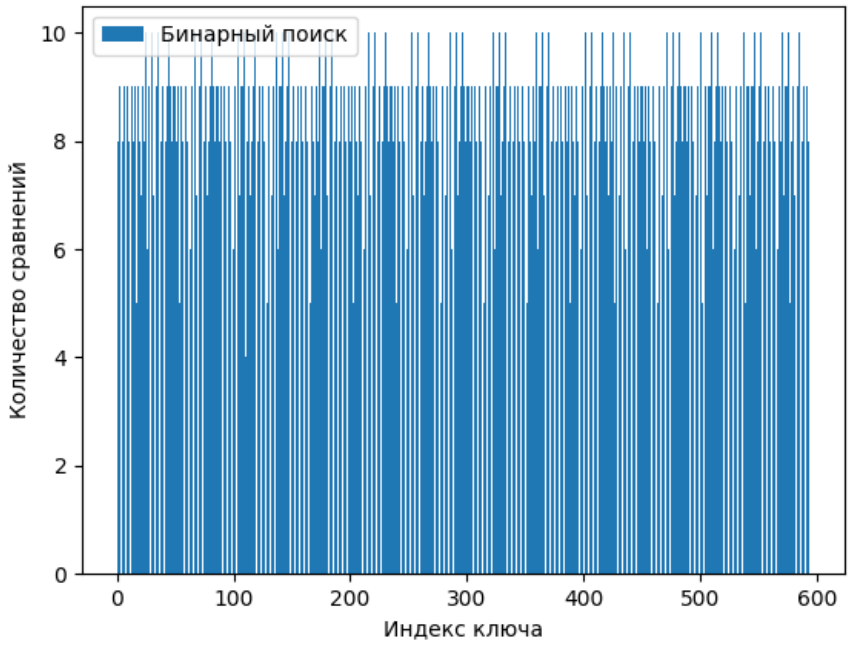

Рис. 4.2: Анализ количества сравнений для двоичного поиска.

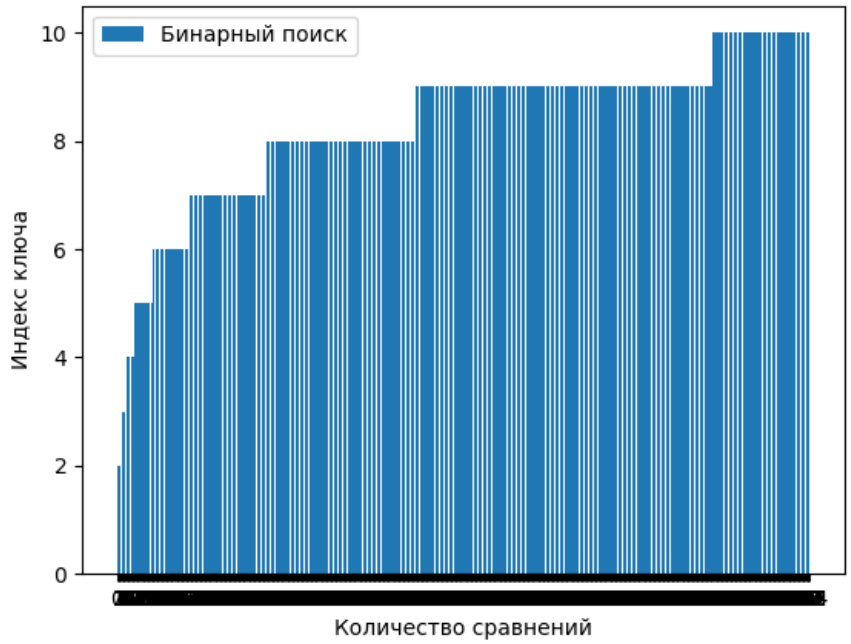

Рис. 4.3: Анализ количества сравнений для двоичного поиска (отсортировано по количеству сравнений).

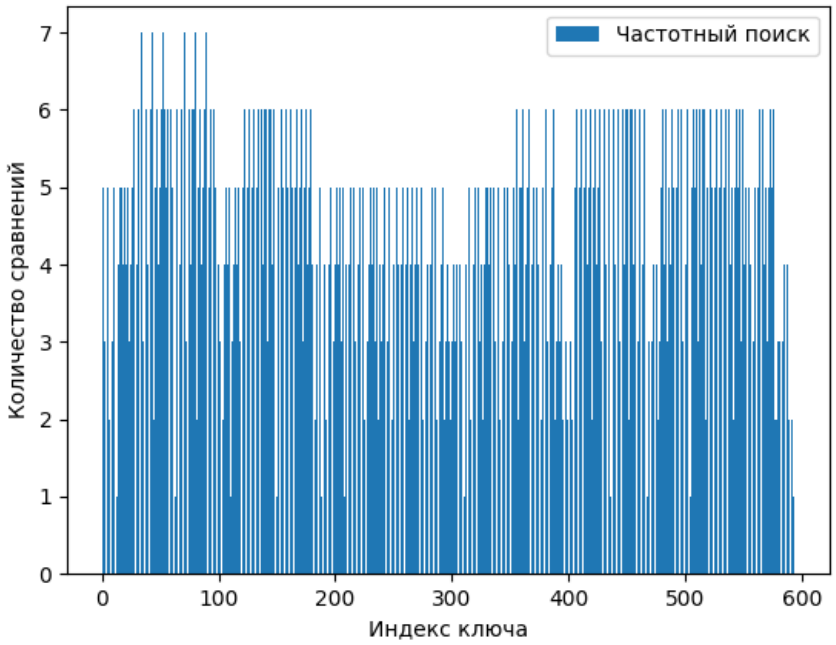

Рис. 4.4: Анализ количества сравнений для поиска с использованием частотного анализа.

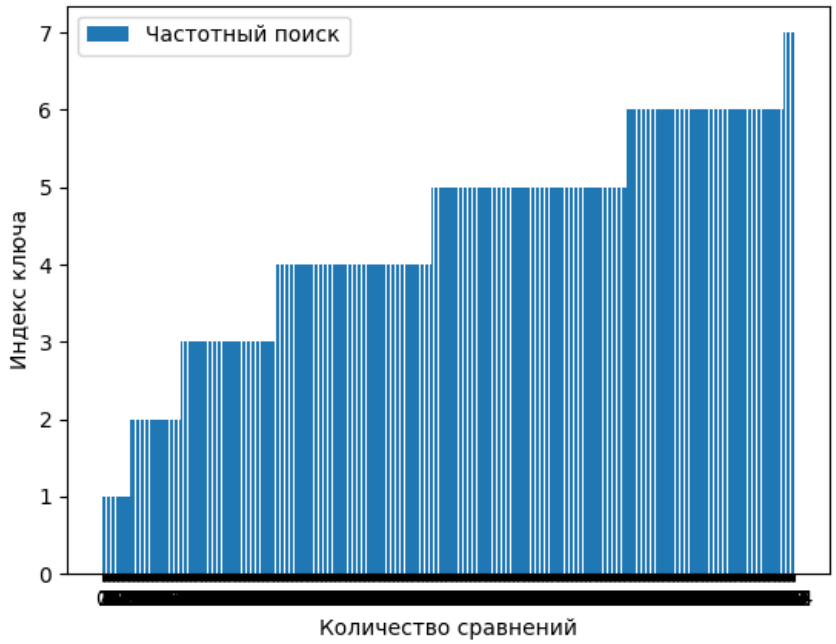

Рис. 4.5: Анализ количества сравнений для поиска с использованием частотного анализа (отсортировано по количеству сравнений).

Вывод

Алгоритм двоичного поиска превосходит алгоритм полного перебора для любово ключа в словаре. Количество сравнений в двоичном поиске не превышает $ceil(\log_2 n)$, где n – размер словаря, когда как в линейном – линейно зависит от n.

Алгоритм поиска с использованием частотного анализа выигрывает у алгоритма двоичного поиска, т. к. уменьшается количество сравнений из-за в заранее известного диапазона поиска. Количество сравнений в поиске с частотным анализом не превышает $ceil(\log_2 max(m))$, где m – размер сегмента.

Минимальное возможное количество сравнений в словаре для всех алгоритмов одинаково и равно 1.

Заключение

В рамках данной лабораторной работы лабораторной работы была достигнута её цель: изучены алгоритмы поиска в словаре. Также выполнены следующие задачи:

- реализованны три алгоритма поиска в словаре;
- замерено количество сравнений во время поиска для всех алгоритмов;
- сделаны выводы на основе проделанной работы.

В результате проведения сравнения количества сравнений алгоритмов, можно сделать вывод, что алгоритм поиска с использованием частотного анализа выигрывает у алгоритма двоичного поиска, т. к. уменьшается количество сравнений из-за в заранее известного диапазона поиска.

Стоит отметить, что и алгоритм двоичного поиска, и алгоритм поиска с использованием частотного анализа требуют предварительной обработки данных (сортировки и сегментации соотвественно), в отличие от алгоритма полного перебора. Но именно это позволяет стать этим алгоритмам намного эффективнее в сравнении с алгоритмом полного перебора, где количество сравнений линейно растёт от размера словаря.

Поставленная цель была достигнута.

Список литературы

[1] Левитин А. В. Глава 4. Метод декомпозиции: Бинарный поиск // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. — С. 180—183. — 576 с.

[2] С.Коутинхо. Введение в теорию чисел. Алгоритм RSA. Москва: Постмаркет, 2001. — 328 с.

[3] Данте Алигьери. Божественная комедия. Перевод М.Лозинского. ББК 84.4 Ит, Д 17, Издательство "Правда М.: 1982, OCR Бычков М.Н.

[4] Руководство по языку C#[Электронный ресурс], - режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/

[5] Visual Studio[Электронный ресурс], - режим доступа: https://visualstudio.microsoft.com/ru/