**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 **09.03.01 Информатика и вычислительная техника**

# ОТЧЕТ

## по лабораторной работе № 7

**Название:** Поиск в словаре

**Дисциплина:** Анализ алгоритмов

Студент ИУ7-52Б _____ Короткая В. М.
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Волкова Л.Л.
(Подпись, дата) (И.О. Фамилия)

Москва, 2021

# Содержание

## Введение

Ассоциативный массив — 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(ключ, значение)
- FIND(ключ)
- REMOVE(ключ)

Предполагается, что ассоциативный массив не может хранить две пары с одинаковыми ключами.

В паре (k, v) значение v называется значением, ассоциированным с ключом k. Где k — это key, а v — value.

Операция FIND(ключ) возвращает значение, ассоциированное с заданным ключом, или некоторый специальный объект UNDEF, означающий, что значения, ассоциированного с заданным ключом, нет. Две другие операции ничего не возвращают (за исключением, возможно, информации о том, успешно ли была выполнена данная операция).

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.

Поддержка ассоциативных массивов есть во многих интерпретируемых языках программирования высокого уровня, таких, как Perl, PHP, Python, Ruby, Tcl, JavaScript и других. Для языков, которые не имеют встроенных средств работы с ассоциативными массивами, существует множество реализаций в виде библиотек.

Примером ассоциативного массива является телефонный справочник: значением в данном случае является совокупность «Ф. И. О. + адрес», а ключом — номер телефона, один номер телефона имеет одного владельца, но один человек может иметь несколько номеров.

Три основных операции часто дополняются другими, наиболее популярные расширения:

- CLEAR — удалить все записи,
- EACH — «пробежаться» по всем хранимым парам,
- MIN — найти пару с минимальным значением ключа,
- MAX — найти пару с максимальным значением ключа.

В последних двух случаях необходимо, чтобы на ключах была определена операция сравнения.

Целью данной лабороторной работы является изучение способа эффекливного по времени и памяти поиска по словарю. Для достижения данной цели необходимо решить следущие задачи:

- исследовать алгоритмы поиска по словарю;
- привести схемы рассматриваемых алгоритмов;
- провести тестирование работы алгоритмов в лучшем, худшем и произвольном случае;
- провести замеры процессорного времени работы алгоритмов поиска по словарю для каждого ключа и для отсутствующего ключа, вывести минимальное, максимальное и среднее время поиска ключа;
- сделать вывод о проделанной работе и описать в отчете.

# 1. Аналитическая часть

В данном разделе приведены теоритические сведения о рассматриваемых алгоритмов.

## 1.1. Алгоритм полного перебора

Идея алгоритма заключается в том, что поиск заданного элемента из множества происходит непосредственно сравниванием каждого элемента этого множества с искомым, до тех пор, пока искомый не найдётся или множество не закончится.

Сложность алгоритма линейно зависит от объёма словаря, а время может стремиться к экспоненциальному времени работы.

## 1.2. Алгоритм бинарного поиска

Данный алгоритм содержит в себе идею, которая заключается в том, что берётся значение ключа из середины словаря и сравнивается с данным. Если значение меньше (в контексте типа данных) данного, то продолжается поиск в левой части словаря, при обратном случае - в правой. На новом интервале также берётся значение ключа из середины и сравнивается с данным. Так продолжается до тех пор, пока найденное значение ключа не будет равно данному.

Поиск в словаре с использованием данного алгоритма в худшем случае будет иметь трудоёмкость $O(log_2 N)$, что быстрее поиска при помощи алгоритма полного перебора. Но стоит учитывать, что алгоритм бинарного поиска работает только для заранее отсортированного словаря.

В случае большого объёма словаря и обратного порядка сортировки, может произойти так, что алгоритм полного перебора будет эффективнее по времени.

## 1.3. Алгоритм частного анализа

Идея алгоритма заключается в составлении частотного анализа. Чтобы провести частотный анализ, необходимо взять первый элемент каждого значения в словаре по ключу и подсчитать частотную характеристику, т.е. сколько раз этот элемент встречается в качестве первого элемента. По полу-

ченным значениям словарь разбивается на сегменты так, что все элементы с одинаковым первым элементом оказываются в одном сегменте.

Далее сегменты упорядочиваются по значению частотной характеристики таким образом, чтобы элементы с наибольшей частотной характеристикой был самый быстрый доступ.

Далее каждый сегмент упорядочивается по значению. Это необходимо для реализации бинарного поиска, который обеспечит эффективный поиск в сегменте при сложности $O(NlogN)$.

## 1.4. Описание словаря

В данной работе словарь представляет собой базу данных имён и имеет вид {key: number, name: string}. Поиск будет реализован по полю key.

## Вывод

В данной работе стоит задача реализации поиска в словаре, были рассмотрены алгоритмы реализации данного поиска.

Входными данными являются:

- словарь записей, вида *username:string, password:string*;
- ключ для поиска по словарю.

Выходными данными является найденая в словаре запись для каждого из реализуемых алгоритмов.

# 2. Конструкторска часть

В данном разделе представленны схемы алгоритмов.

## 2.1. Схемы алгоритмов

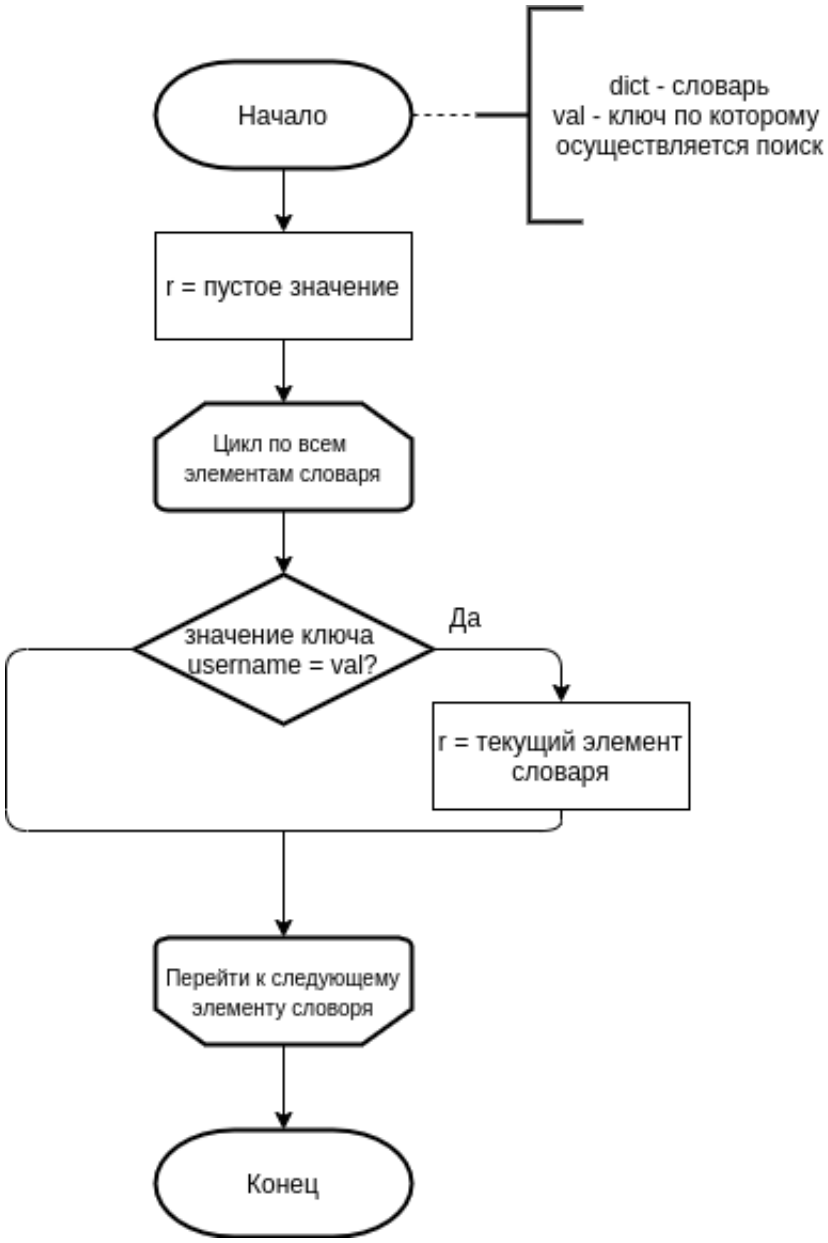


Рис. 2.1: Схема алгоритма полного перебора.

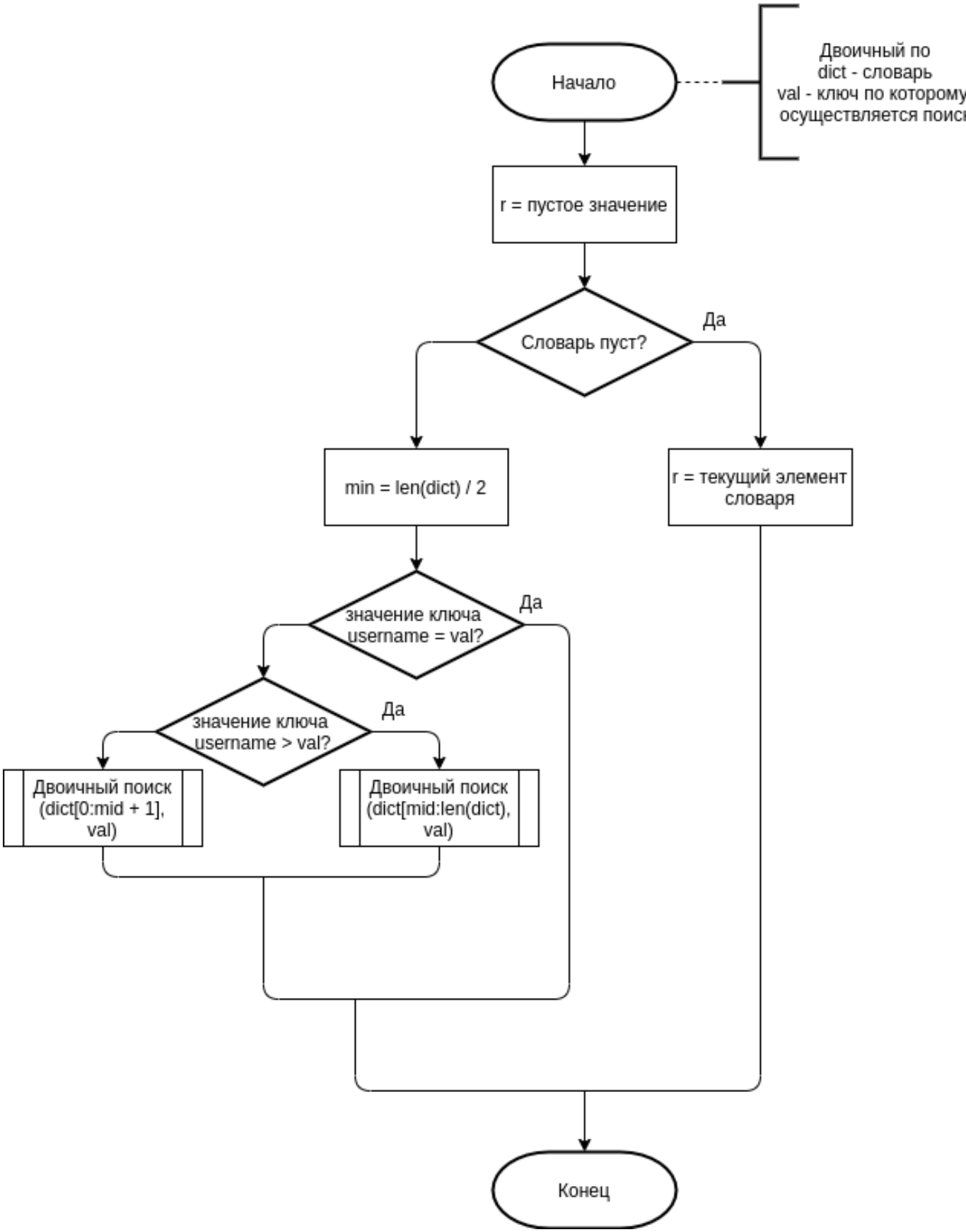


Рис. 2.2: Схема алгоритма с бинарным поиском.

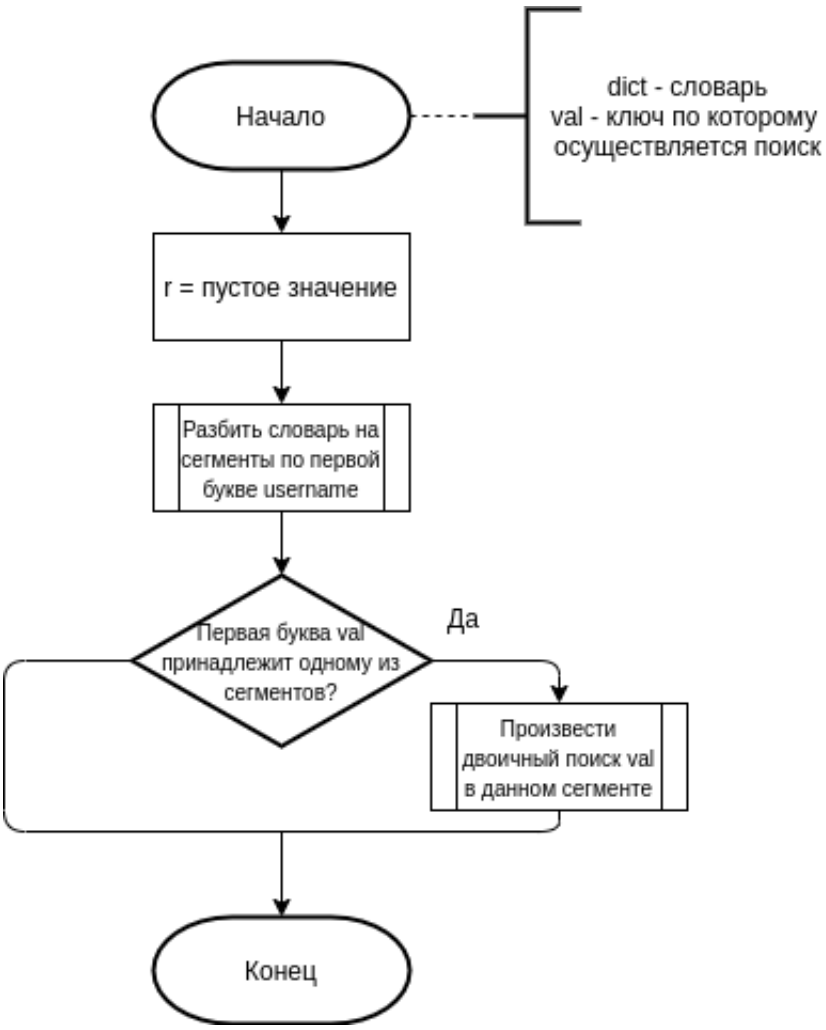


Рис. 2.3: Схема алгоритма с частотным анализом.

## 2.2. Описание структуры программного обеспечения

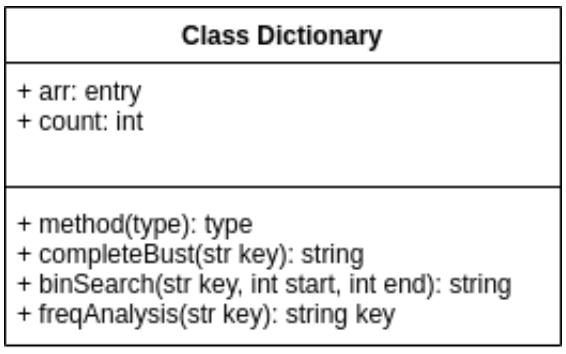


Рис. 2.4: Структура ПО.

## 2.3. Описание структур данных

Для реализации данных алгоритмов, введем некоторые типы данных:

- entry - тип данных описывающий пару ключ(string) + значение(string);
- segment - тип данных описывающий сегмент частотного анализа первая буква(char) + массив(int) с индексами ключей начинающимися на эту букву

## Вывод

На основе теоритических данных, полученных из аналитического раздела, были построены схемы реализируемых алгоритмов.

Так же, было приведено описание вводимых типов данных.

# 3. Технологическая часть

В данном разделе приведены средства реализации, требования к ПО и листинги кода.

## 3.1. Средства реализации

В качестве языка программирования был выбран c++. Данный язык знаком и предостовляет все необходимые ресурсы. В качестве среды разработки я использовала Visual Studio Code, т.к. считаю его достаточно удобным и легким. Visual Studio Code подходит не только для Windows, но и для Linux, это еще одна причина, по которой я выбрала VS code, т.к. у меня установлена ОС fedora 34.

## 3.2. Сведенья о модулях программы

Данноя программа разбита на модули:

- main.cpp - файл, содержащий точку входа в программу;
- dictionary.cpp - файл, содержащий реализацию алгоритмов поиска в словаре.

## 3.3. Реализация конвейера

Листинг 3.1: Созданные типы данных.

```
1 struct entry{
2     std::string value;
3     std::string key;
4 } ;
5
6 struct segment{
7     char first;
8     std::vector<int> arrInt;
9 };
```

Листинг 3.2: Функция реализация алгоритма полного перебора.

```
std::string Dictionary::completeBust(std::string key){
    std::string val = "meh";
    for (int i = 0; i < count; i++){
        if (key == arr[i].key)
            val = arr[i].value;
    }
    return val;
}
```

Листинг 3.3: Функция реализация бинарного поиска.

```
std::string Dictionary::binSearch(std::string key, int start,
    std::string val = "meh";
    if (end - start < 2)
    {
        if (key == arr[start].key)
            return arr[start].value;
        if (key == arr[end].key)
            return arr[end].value;
        return val;
    }
    int mid = (end - start)/2;
    if (key == arr[mid + start].key)
        return arr[mid + start].value;
    if (key > arr[mid + start].key)
        return binSearch(key, start + mid, end);
    else
        return binSearch(key, start, start + mid);
}
```

Листинг 3.4: Функция разделения на сегменты.

```
void Dictionary::createSeg(){
    int f;
    for (int i = 0; i < count; i++)
    {
```

```
5           f = arr[i].key[0] - '0';
6           seg[f].arrInt.push_back(i);
7       }
8   }
```

Листинг 3.5: Функция реализации частотного анализа.

```
1   std::string Dictionary::freqAnalysis(std::string key){
2       int f = key[0] - '0';
3       int size = seg[f].arrInt.size();
4       return binSearch(key, seg[f].arrInt[0],
5                                seg[f].arrInt[size - 1]);
6       return "meh";
7   }
```

## 3.4. Тестирование

В данном разделе будет приведена таблица с тестами (таблица 3.1). Тестирование проводилось в словаре, содержащем следущие записи:

- key: "номер ОМС"+ value: "ФИО".

Таблица 3.1: Таблица тестов

| Входные данные | Пояснение | Результат |
|---|---|---|
| 0002530407440900 | Первый элемент | Ответ верный |
| 5091479945279275 | Средний элемент | Ответ верный |
| 9997951582572083 | Последний элемент | Ответ верный |
| 2123566581544639 | Произвольный элемент | Ответ верный |
| 25436523 | Несуществующий элемент | Ответ верный |

Все тесты пройдены.

## Вывод

В данном разделе были реализованны вышеописанные алгоритмы поиска в словаре. Было разработано ПО, удовлетворяющее предьявляемым требованиям. Так же были представлены соответствующие листинги с кодом программы. А так же проведено тестирование разработанного ПО.

# 4. Исследовательская часть

В данном разделе будет произведено измерение временных характеристик.

## 4.1. Технические характеристики

Технические характеристики устройства на котором выполнялось исследование:

- процессор Intel® Core™ i5-10210U CPU @ 1.60GHz × 8;
- память 15.3 GiB;
- операционная система Fedora 34 (Workstation Edition) 64-bit.

## 4.2. Временные характеристики

Так как поиск в словаре считается короткой задачей, воспользуемся усреднением массового эксперимента. Для этого сложим результат работы алгоритма n раз (n >= 10), после чего поделим на n. Тем самым получим достаточно точные характеристики времени. Сравнение произведем при n = 1000.

| 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 |

Рис. 4.1: Время работы алгоритмов.

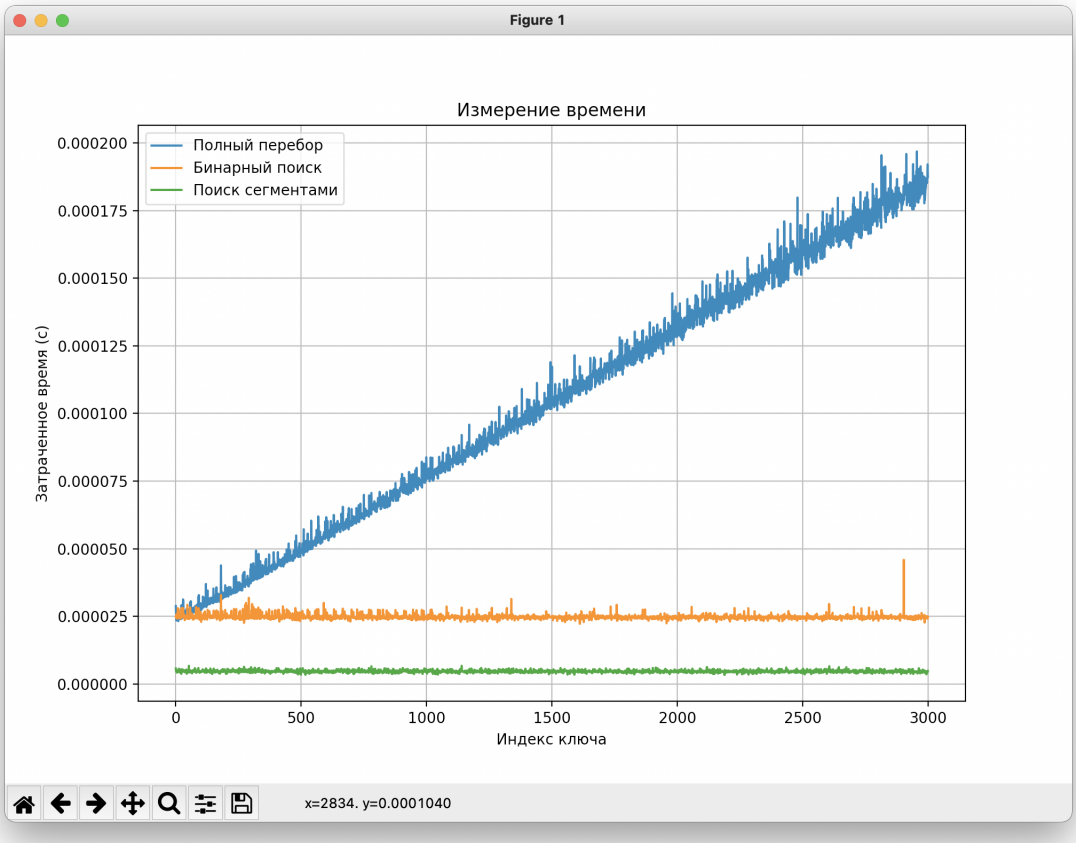


Рис. 4.2: Зависимость времени паботы алгоритмов поиска от индекса клюса.

- представлен результат поиска первого значения. Выигрыш алгоритма поиска полным перебором обосновывается тем, что он тратит лишь одно сравнение для того, чтобы найти первый ключ, в том время, когда бинарный поиск затрачивает гораздо больше сравнений. Частичный анализ работает чуть медленнее, так как ему нужно произвести дополнительное сравнение первых букв.

- представлен результат поиска среднего значения. Выйгрыш бинарного поиска в том что он находит середину за одну итерацию.

- приведено сравнение времени выполнения трех алгоритмов для поиска последнего значения. По результатам эксперимента видно, что поиск полным перебором затрачивает больше всего времени, т.к. он последовательно обходит все элементы, в то время, как остальные два алгоритма выполняют поиск значительно быстрее.

- представлен результат поиска произвольного ключа. Алгоритм полного

перебора работает медленнее всех.

- произведено аналогичное сравнение, только в качестве искомого ключа взят несуществующий. Аналогично поиск полным перебором затрачивает больше всего времени по вышеописанной причине.

## 4.3. Колличество 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).

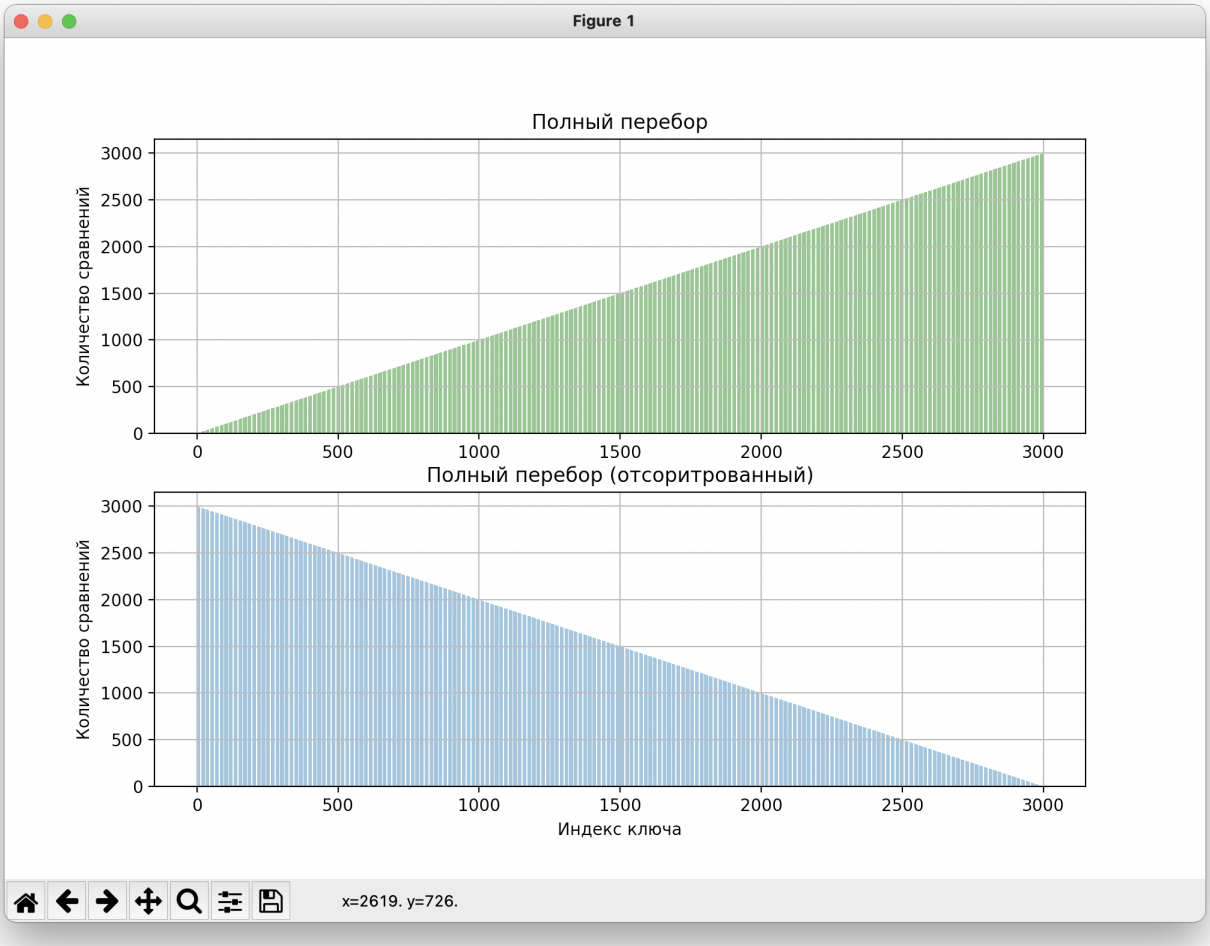


Рис. 4.3: Кол-во сравнений при поиске ключа в словаре (полным перебором)

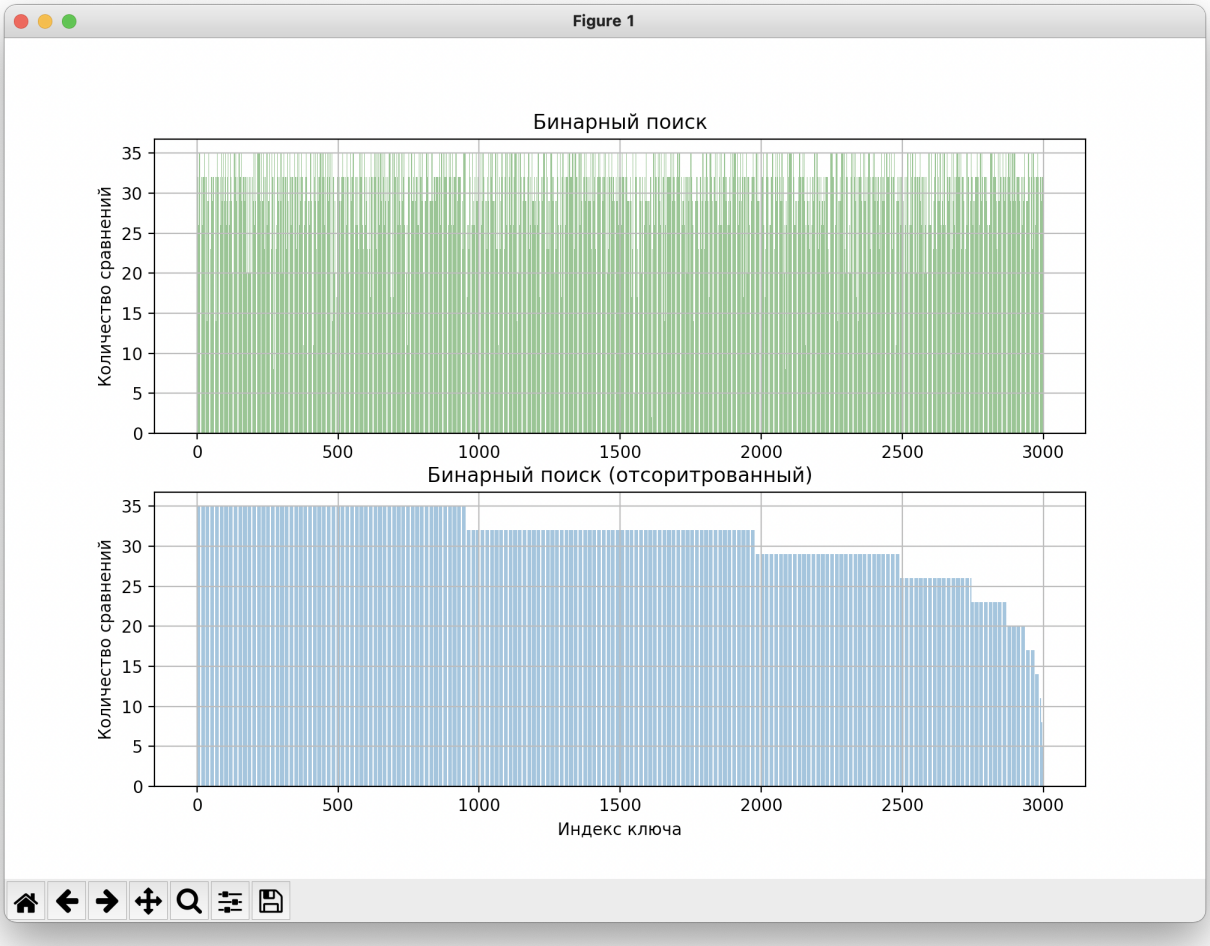


Рис. 4.4: Кол-во сравнений при поиске ключа в словаре (бинарным поиском)

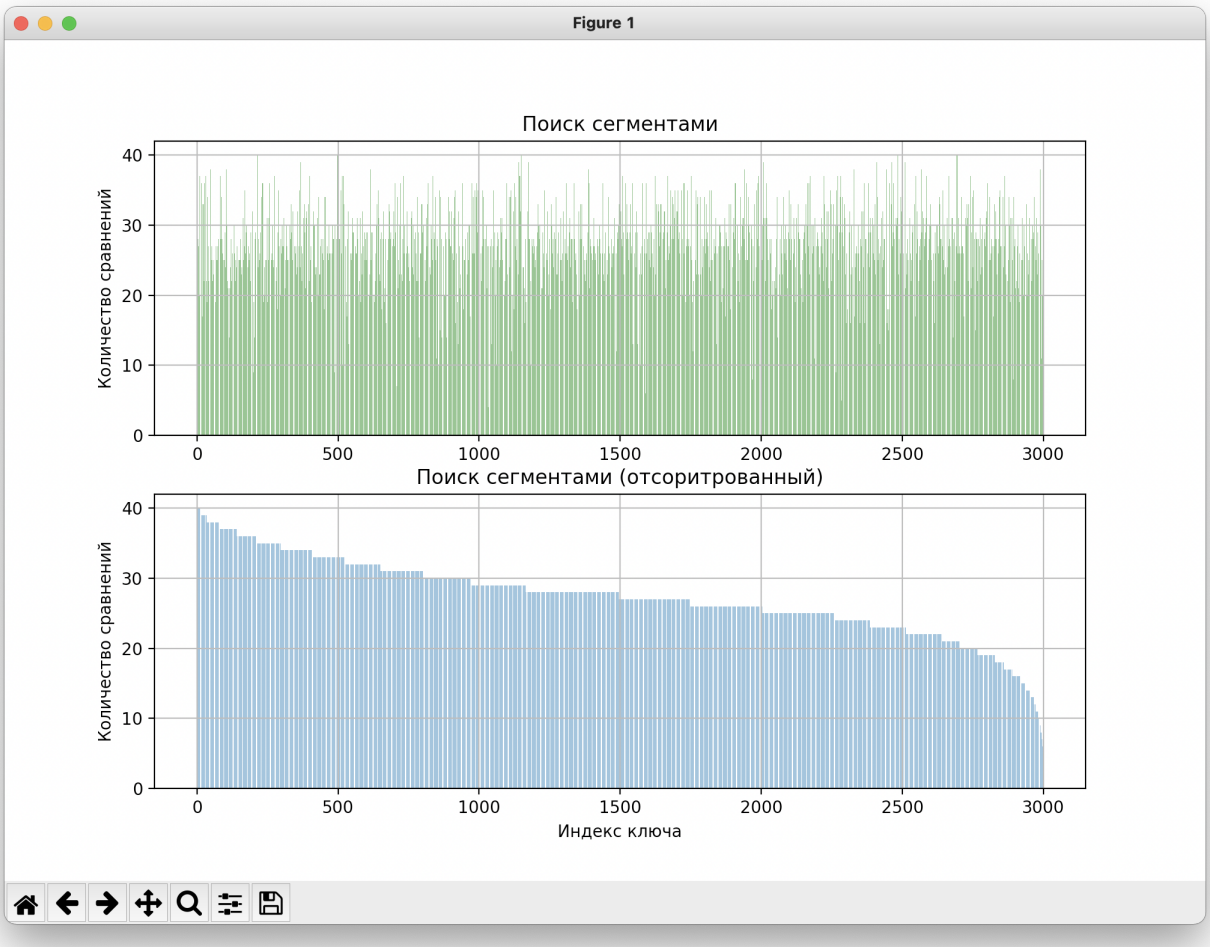


Рис. 4.5: Кол-во сравнений при поиске ключа в словаре (поиском сегментами)

## 4.4. Вывод

В данном разделе было произведено сравнение трех алгоритмов. По результатам исследования было доказано, что алгоритм полного перебора в основном работает медленнее всех, за исключением, когда ключ лежит достаточно близко к началу.

## Заключение

В данной лабораторной работе были изучены алгоритмы поиска по ассоциативному словарю.

Среди рассмотренных алгоритмов наиболее эффективным по времени является алгоритм бинарного поиска. Однако при больших размерностях входных массивов алгоритм бин поиска становиться менее эффективным, чем алгоритм частотного анализа. Поэтому, при большом числе элементов входного массива стоит использовать данный алгоритм.

В рамках лабораторной работы цель достигнута и выполнены следующие задачи:

- исследованы алгоритмы поиска по словарю;
- приведены схемы рассматриваемых алгоритмов;
- проведены тестирование работы алгоритмов в лучшем, худшем и произвольном случае;
- проведены замеры процессорного времени работы алгоритмов поиска по словарю для каждого ключа и для отсутствующего ключа, вывести минимальное, максимальное и среднее время поиска ключа;
- сделан вывод о проделанной работе и описать в отчете.

## Список литературы


1. Дж. Макконнел. Анализ алгоритмов. Активный обучающий подход. – М.: Техносфера, 2017. – 267с.

2. Основы программирования на языках Си и C++ для начинающих[Электронный ресурс]. Режим доступа: http://cppstudio.com/ (дата обращения 10.10.2021)

3. LINUX.ORG.RU - Русскоязычная информация о ОС Linux[Электронный ресурс] Режим доступа://www.linux.org.ru/(дата обращения 25.10.2021)

4. Документация языка C++ 98 [Электронный ресурс], режим доступа: http://www.open-std.org/JTC1/SC22/WG21/ (дата обращения 10.12.2021)

5. Кнут Д. Э. Искусство программирования. Том 3. Сортировка и поиск = The Art of Computer Programming. Volume 3. Sorting and Searching / под ред. В. Т. Тертышного (гл. 5) и И. В. Красикова (гл. 6). — 2-е изд. — Москва: Вильямс, 2007. — Т. 3. — 832 с. — ISBN 5-8459-0082-1.