**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** Чаушев А.К..

**Группа** ИУ7-56Б

**Оценка (баллы)** _____

**Преподаватели** Волкова Л.Л., Строганов Ю.В.

Москва — 2020 г.

# Введение

Поиск слово в словаре [2] – задача встречающаяся во всех областях программирования.

Цель данной работе является изучение алгоритмов решения данной задачи.

Задача: реализовать и провести анализ алогоритмов полночо перебора, бинарного поиска, и поиска на основе сегментов.

# 1 Аналитическая часть

В данном разделе приведено описание алгоритма алгоритмов поиска.

## 1.1 Определение словаря

Словарь это ассоциативный массив – абстрактный тип данных, хранящий данные в формате пар ключ – значения, поддерживающий операции добавление пары, поиска, удаление пары по ключу. Ключи ассоциативного массива уникальны,в то время как значения могут повторятся

## 1.2 Поиск по словарю

Операция поиска по словарю должна принимать в качестве входного параметра ключ и возвращать значение, с ним ассоциируемое. Если такого значения не существует возвращается объект об этом сигнализирующий.

## 1.3 Алгоритм полного перебора

Полный перебор подразумевает проверку всех возможных вариантов ответа. Для поиска в словаре полный перебор итерируется по ключам словаря и сравнивает их с искомым, пока не найдёт нужный или не переберёт все ключи. Данный алгоритм подробно рассмотрен в [2].

## 1.4 Алгоритм бинарного поиска

Бинарный поиск - алгоритм поиска элемента в отсортированном массиве, который заключается в делении части массива, которой точно принадлежит искомой элемент, на две равные части и определения в какой из частей находится элемент путём сравнения его с серединным элементом. Такие разбиения происходят, пока не будет найден нужный элемент или область поиска не опустеет. Для поставленной задачи необходимо искать ключ в отсортированном массиве ключей.

## 1.5 Алгоритм поиска на основе сегментов

Алгоритм поиска в словаре на основе сегментов является оптимизированным алгоритмом поиска в словаре. Алгоритм требует предварительного разбиения ключей исходного словаря на непересекающиеся подмножества - сегменты. Каждый из ключей исходного словаря входит ровно в один сегмент. На первом этапе происходит поиск сегмента, которому принадлежит заданный ключ. Это значительно сужает область дальнейшего поиска. Затем происходит поиск заданного ключа в сегменте. Разбиение на сегменты происходит на основе каких-то их общих свойств. Например, если все ключи содержат один из трёх возможных префиксов, то необходимо разбить их на три сегмента, в каждом из которых будут все ключи, содержащие конкретный префикс.

## 1.6 Вывод

В данном разделе были расммотрены особенности и определение ассоциативного массива и принципы основных основных алгоритмов поиска в словаре.

# 2 Конструкторская часть

Рассмотрим алгоритм поиска.

## 2.1 Схемы алгоритмов

На рисунке 1 изображена схема алгоритма полного перебора.

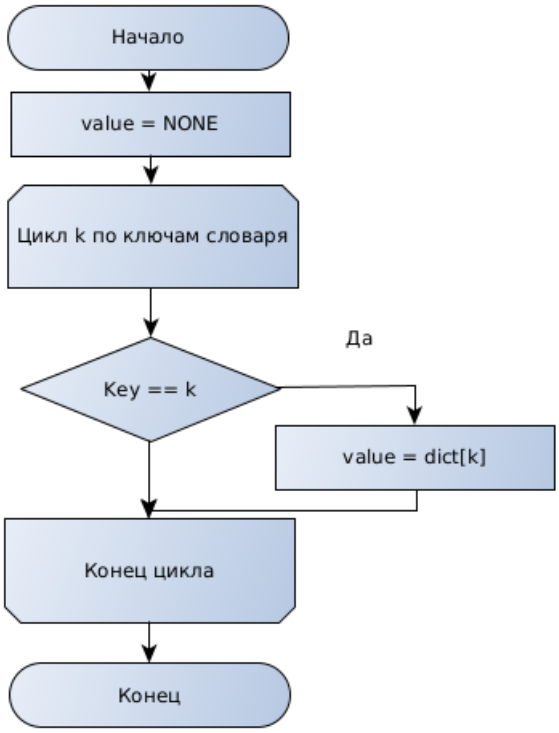


Рис. 1 – Схема алгоритма полного перебора.

На рисунке 2 представлена схема алгоритма бинарного поиска.

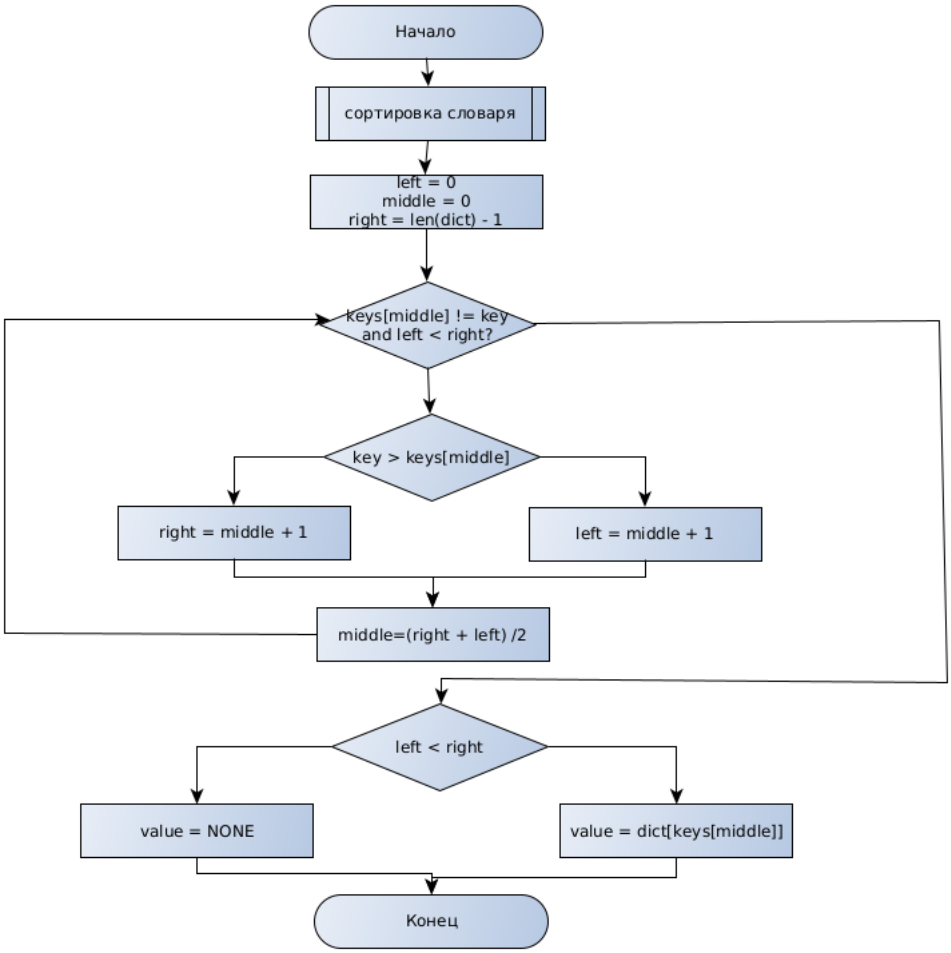


Рис. 2 – Схема алгоритма бинарного поиска.

На рисунке 3 представлена схема алгоритма бинарного поиска с помощью сегментации.

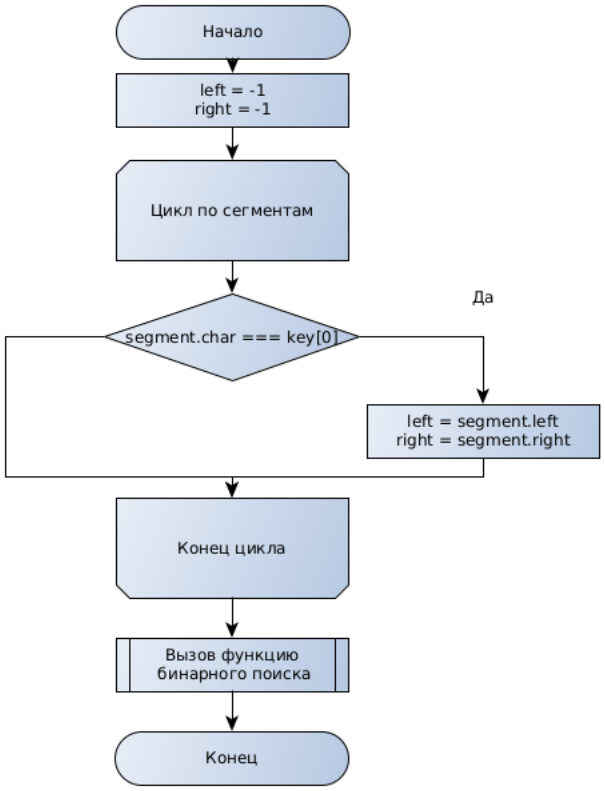


Рис. 3 – Схема алгоритма бинарного поиска с помощью сегментации.

## 2.2 Трудоёмкости алгоритмов

Для задачи поиска в ассоциативном массиве можно выделить следующие случам:

- лучший - искомый клюя найден при первом сравнении;
- произвольный;
- худший - искомый ключ найден при последнем сравнении или отсутствует.

Пусть $k0$ - количество операций на старте, $k1$ - количество операций при каждом сравнении, $N$ - разсерность словаря.

### 2.2.1 Полный перебор

Лучший случай - $k0 + k1$.
Худший случай - $k0 + N * k1$.
Средний случай - $ko + k1 * (1 + N/2 - 1/(N + 1))$.

### 2.2.2 Двоичный поиск

В процессе двоичного поиска на каждой итерации область поиска уменьшается в два раза. Обход словаря можно рассматривать как обход двоичного дерева.
Лучший случай - $k0 + k1$.
Худший случай - $k0 + k1 * log(N)$.

### 2.2.3 Поиск на основе сегментов

Пусть N1, N2 - количества сегментов первого и второго уровней соответственно; pi, pj - вероятности нахождения ключа в i-ом сегменте первого уровня и в j-ом сегменте второго уровня, соответственно, а F1, F2 -поиск сегмента сооветствующего уровня. Средний случай - $\sum_{i=0}^{N} 1(F1 + \sum_{j=0}^{N} 2(F2) * pj) * pi$.

# 3 Вывод

В данном разделе были расммотрены схемы алгоритмы и приведены оценки их трудоёмкостей. ...

# 4 Технологическая часть

В этом разделе будет обоснован выбор языка программирования, приведёны реализации алгоритмов и проведено их тестирование.

## 4.1 Средства реализации

В качестве языка программирования был выбран `Kotlin`. Данный язык имеет полную совместимость с Java. Как и Java, C и C++, Kotlin[1] — это статически типизированный язык. Он поддерживает как объектно-ориентированное, так и процедурное программирование. Программа, написанная на `Kotlin`, будет доступна на всех платформах.

## 4.2 Методы замера времени в программе

### 4.2.1 Время

Время замерялось с помощью функции `measureTimeMillis`, которая измеряет процессорное время в милисекундах.

```
1 inline fun measureTimeMillis(block: () -> Unit): Long
```

Тестирование проводится на процессоре с количеством логических потоков равным 4.

### 4.3 Реализация алгоритмов

В листингах 1 , 2 и 3 представлены реализации алгоритмов полного перебора, бинарного поиска и поиска на основе сегментов, соответственно.

Листинг 1 – функция полного перебора

```
    fun get(key: String): V? {
        entries.forEach { entry ->
            if (entry.key == key) {
                return entry.value
            }
        }

        return null
    }
```

Листинг 2 – функция бинарного поиска

```
     private fun binarySearch(
     sortedList: List<Entry<String, V>>,
     key: String,
     min: Int,
     max: Int): V? {
        var left = min
        var middle = min
        var right = max

        while (sortedList[middle].key.compareTo(key) != 0 && left < right) {
            if (key > sortedList[middle].key) {
                left = middle + 1
            } else {
                right = middle - 1
            }

            middle = (left + right) / 2
        }

        if (left > right) {
            return null
        }

        return sortedList[middle].value
    }
```

Листинг 3 – поиск по сегментам

```
    fun segmentSearch(key: String): V? {
        var left = -1
        var right = -1
        for (segment in segments) {
            if (segment.char === key[0]) {
                left = segment.left
                right = segment.right
                break
            }
        }
        var sortedList = entries.sortedBy { it.key }

        return binarySearch(sortedList, key, left, right);
    }
```

# 5 Экспериментальная часть

В этом разделе будет исследована временная эффективность реализованных алгоритмов.

## 5.1 Примеры работ

На рисунке 4 изображены примеры работ.

```
Loading Dictionary...
Dictionary loaded
Enter email to get user id: susan12614@hotmail.com
73
```

Рис. 4 – Успешное выполнение

## 5.2 Замеры времени

Для каждого из реализованных алгоритмов были произведены замеры времени выполнения поиска каждого из ключей словаря, а также ключа, отсутствующего в словаре. Замеры были выполнены на 300 итерациях ддля каждого ключа, после чего результирующее время усреднено по всем итерациям. Результаты замеров показаны на рисунке 5.

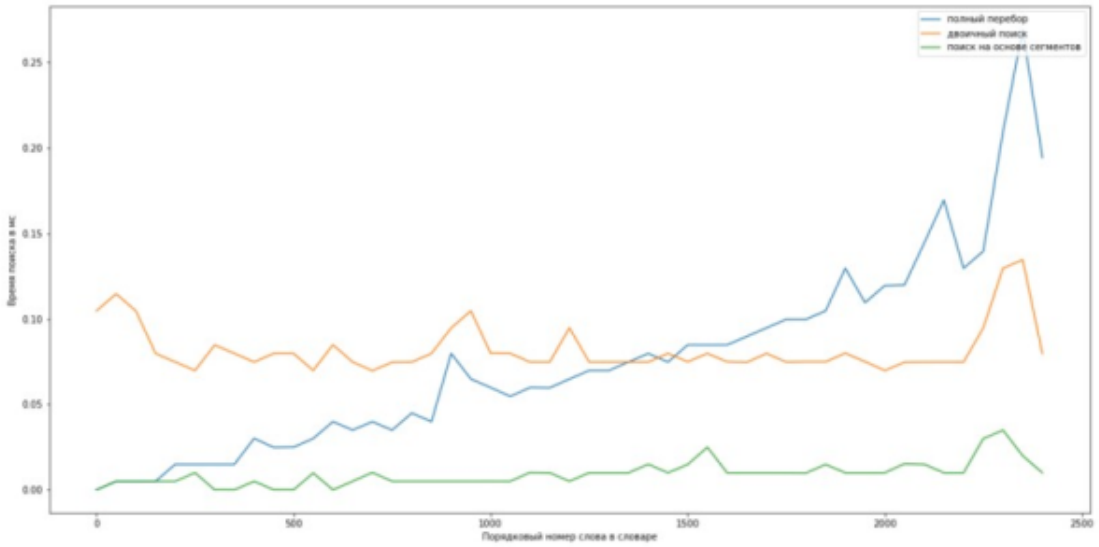


Рис. 5 – Среднее время выполнения алгоритмов

Наилучшую временную эффективность показала реализация поиска на основе сегментов. В среднем данный алгоритм эффективней бинарного поиска почти в 9 раз. Бинарный поиск эффективнее полного перебора на ключах, которые распологаются дальше середины массива. На ключах близких к началу массива ключей полный перебор оказывается эффективней двоичного поиска.

## 5.3 Выводы

В данном разделе эксперементальным путём был найден наиболее эффективный алгоритм поиска в словаре - поиск на основе сегментов.

# Заключение

В ходе лабораторной работы были реализованы и исследованы алгоритмы поиска по ключу в ассоциативном массиве, выполнена оценка трудоёмкости реализованных алгоритмов, эксперементально потдтверждено различие эффективности алгоритмов.

При выполнения лабораторной работы цель была достигнута и выполнены все задачи.

# Список литературы


[1] JetBrains Kotlin [ЭЛ. РЕСУРС] Режим доступа: https://kotlinlang.org/. (дата обращения: 21.10.2020).

[2] Словари в Python: [ЭЛ. РЕСУРС] Режим доступа:https://foxford.ru/wiki/informatika/slovari-assotsiativnyemassivy-v-python. (дата обращения: 19.12.2020).