

	<p><b>Министерство образования и науки Российской Федерации</b> <b>Федеральное государственное бюджетное образовательное</b> <b>учреждение</b> <b>высшего образования</b> <b>«Московский государственный технический университет</b> <b>имени Н.Э. Баумана</b> <b>(национальный исследовательский университет)»</b> <b>(МГТУ им. Н.Э. Баумана)</b></p>
---	--

---

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе № 7

По курсу "Анализ Алгоритмов"

## Поиск в словаре

Студент:

Турсунов Жасурбек Рустамович

Группа: ИУ7-56Б

Преподаватели:

Волкова Лилия Леонидовна  
Строганов Юрий Владимирович

Москва, 2020 г.

# Содержание

<b>Введение</b>	<b>2</b>
<b>1 Аналитическая часть</b>	<b>3</b>
1.1 Алгоритм полного перебора . . . . .	3
1.2 Алгоритм двоичного поиска . . . . .	3
1.3 Алгоритм частотного анализа . . . . .	4
1.4 Описание словаря . . . . .	4
1.5 Вывод . . . . .	5
<b>2 Конструкторская часть</b>	<b>6</b>
2.1 Разработка алгоритмов . . . . .	6
2.2 Вывод . . . . .	8
<b>3 Технологическая часть</b>	<b>9</b>
3.1 Требования к программному обеспечению . . . . .	9
3.2 Средства реализации . . . . .	9
3.3 Листинг кода . . . . .	10
3.4 Тестирование функций . . . . .	11
3.5 Вывод . . . . .	11
<b>4 Исследовательская часть</b>	<b>12</b>
4.1 Системные характеристики . . . . .	12
4.2 Пример работы . . . . .	12
4.3 Сравнительный анализ на основе замеров времени работы программы . . . . .	13
4.4 Вывод . . . . .	13
<b>Заключение</b>	<b>14</b>
<b>Список литературы</b>	<b>15</b>

# Введение

Целью данной лабораторной работы является изучение способа эффективного по времени и по памяти поиска по словарю.

Словарь, или 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.[1]

В паре  $(k, v)$  значение  $v$  называется значением, ассоциированным с ключом  $k$ . Где  $k$  – это *key*, а  $v$  – *value*. Семантика и названия вышеупомянутых операций в разных реализациях ассоциативного массива могут отличаться.

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 ходе лабораторной предстоит выполнить следующие задачи:

1. рассмотреть и изучить алгоритмы полного перебора, двоичного поиска и эффективного поиска по словарю;
2. сравнить временные характеристики каждого из рассмотренных алгоритмов;
3. на основе проделанной работы сделать выводы.

# 1 Аналитическая часть

В данной части будут представлены теоретические сведения о рассматриваемых алгоритмах.

## 1.1 Алгоритм полного перебора

Алгоритмом полного перебора называют метод решения задачи, при котором по очереди рассматриваются все возможные варианты исходного набора данных. В случае словарей будет произведен последовательный перебор элементов словаря до тех пор, пока не будет найден необходимый. Сложность такого алгоритма зависит от количества всех возможных решений, а время решения может стремиться к экспоненциальному времени работы.

Пусть алгоритм нашёл элемент на первом сравнении, тогда, в лучшем случае, будет затрачено  $k_0 + k_1$  операций, на втором –  $k_0 + 2k_1$ , на последнем –  $k_0 + Nk_1$ . Тогда средняя трудоёмкость может быть рассчитана по формуле 1, где  $\Omega$  – множество всех возможных случаев.

$$\begin{aligned} \sum_{i \in \Omega} \rho_i \cdot f_i &= (k_0 + k_1) \frac{1}{N+1} + (k_0 + 2k_1) \frac{1}{N+1} + (k_0 + 3k_1) \frac{1}{N+1} + (k_0 + Nk_1) \frac{1}{N+1} + \\ &+ (k_0 + Nk_1) \frac{1}{N+1} = k_0 \frac{N+1}{N+1} + k_1 + \frac{1 + 2 + \dots + N + N}{N+1} = k_0 + k_1 \left( \frac{N}{N+1} + \frac{N}{2} \right) = \\ &= k_0 + k_1 \left( 1 + \frac{N}{2} - \frac{1}{N+1} \right) \quad (1) \end{aligned}$$

## 1.2 Алгоритм двоичного поиска

Алгоритм двоичного поиска применяется к заранее упорядоченному словарю.[2] Процесс двоичного поиска можно описать при помощи шагов:

1. получить значение ключа, находящееся в середине словаря, и сравнить его с данным;
2. в случае, если значение меньше (в контексте типа данных) данного, продолжить поиск в левой части словаря, в обратном случае – в правой;
3. на новом интервале получить значение ключа из середины этого интервала и сравнить его с данным;

4. продолжать поиск до тех пор, пока найденное значение ключа не будет равно данному.

Поиск в словаре с использованием данного алгоритма в худшем случае будет иметь трудоемкость  $O(\log_2 N)$ , что быстрее поиска при помощи алгоритма полного перебора. Но стоит учитывать тот факт, что данный алгоритм работает только для заранее упорядоченного словаря. В случае большого объема данных и обратного порядка сортировки может произойти так, что алгоритм полного перебора будет эффективнее по времени, чем алгоритм двоичного поиска.

### 1.3 Алгоритм частотного анализа

Алгоритм частотного анализа на вход получает словарь и на его основе составляется частотный анализ. Чтобы провести частотный анализ нужно 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 полученным значениям словарь разбивается на сегменты так, что все элементы с одинаковым первым элементом оказываются в одном сегменте.

Сегменты упорядочиваются по значению частотной характеристики так, чтобы к 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  $O(\log_2 N)$ .

Таким образом, сначала выбирается нужный сегмент, а затем в нем проводится бинарный поиск нужного элемента. Средняя трудоёмкость при длине алфавита  $M$  может быть рассчитана по формуле 2.

$$\sum_{i \in [1, M]} (f_{select} + f_{search}) + \rho_i \quad (2)$$

### 1.4 Описание словаря

Словарь, реализованный в данной работе, имеет вид  $id : number, email : string$ , что представляет собой базу данных о пользователях с их уникальной почтой.

## 1.5 Вывод

Были рассмотрены разные алгоритмы поиска в словаре. Дана предварительная оценка времени для каждого алгоритма.

## 2 Конструкторская часть

В данном разделе представлены схемы рассматриваемых алгоритмов.

### 2.1 Разработка алгоритмов

На рисунках 1, 2 и 3 приведены схемы алгоритмов поиска в словаре перебором, двоичным поиском и частотным анализом соответственно.

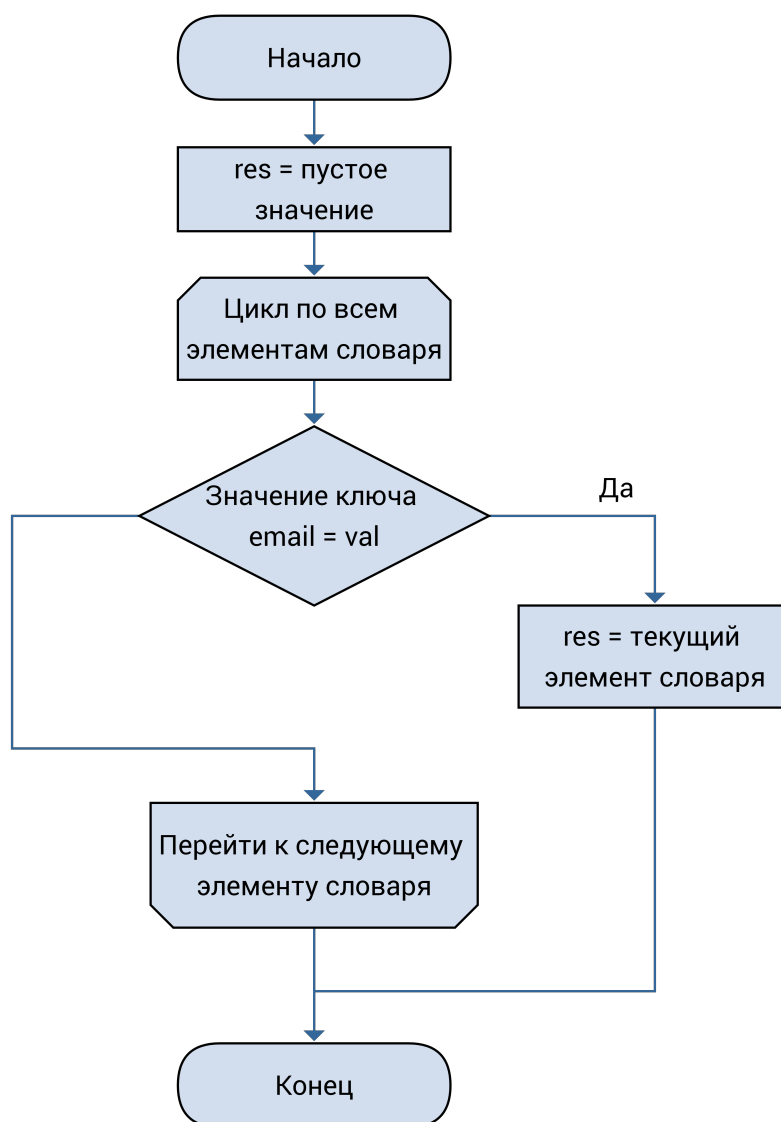


Рис. 1: Схема алгоритма полного перебора в словаре.

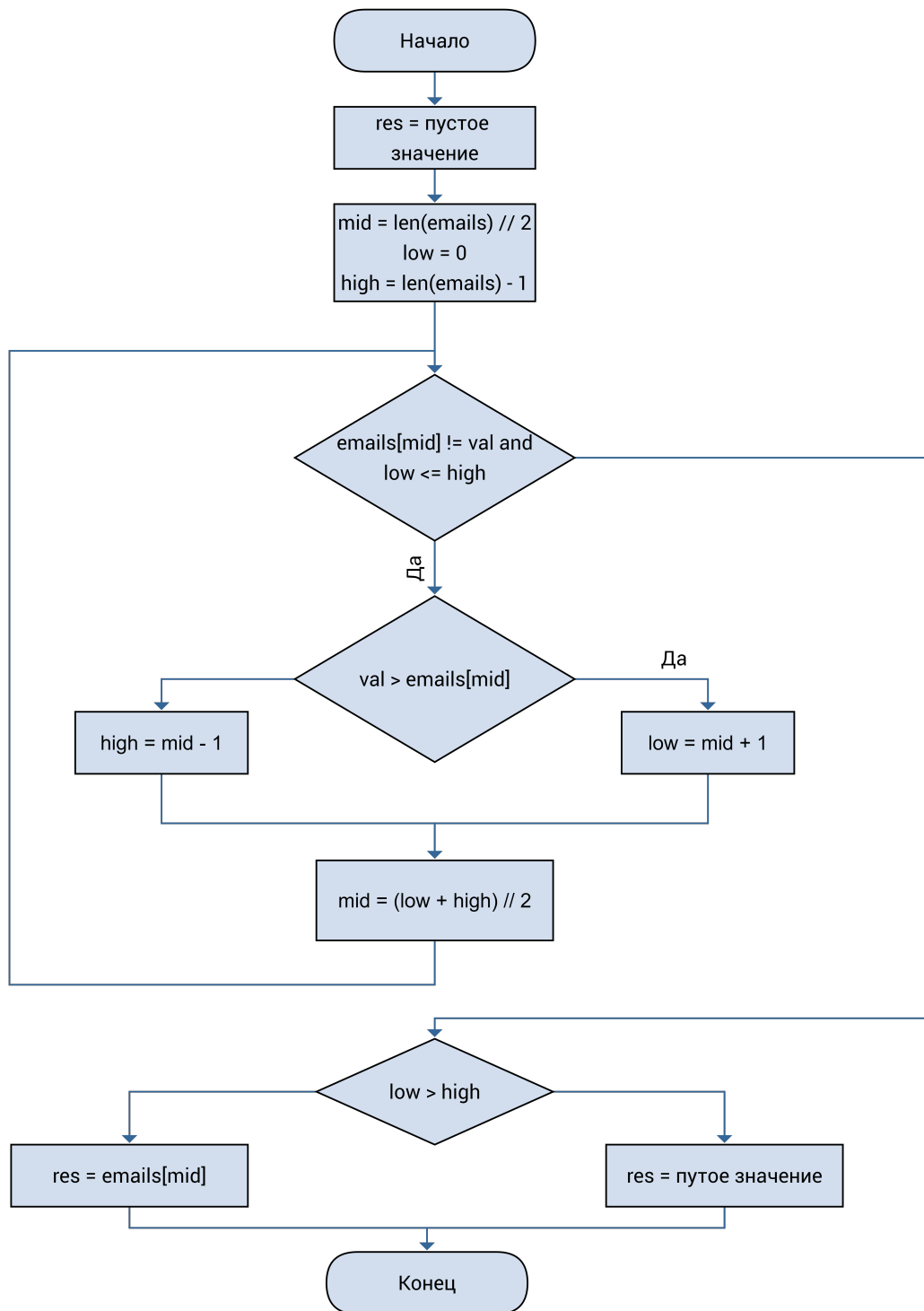


Рис. 2: Схема алгоритма бинарного поиска в словаре.



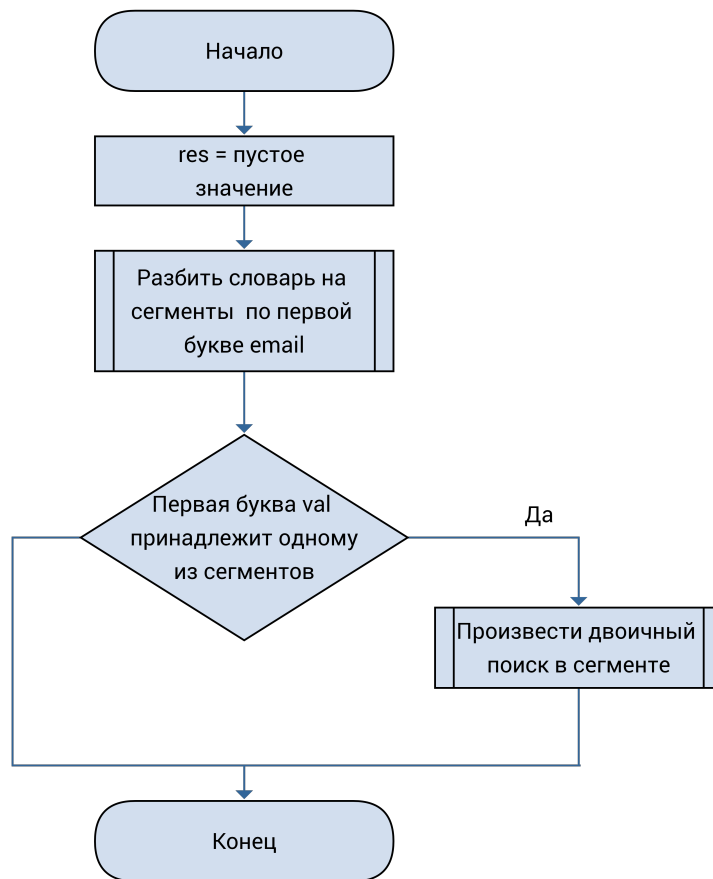


Рис. 3: Схема алгоритма комбинированного поиска в словаре.

## 2.2 Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы требуемых алгоритмов.

## **3 Технологическая часть**

В данном разделе будут рассмотрены требования к программному обеспечению, средства реализации и представлен листинг кода.

### **3.1 Требования к программному обеспечению**

1. на вход подается словарь и значение ключа email;
2. на выходе - элемент словаря.

### **3.2 Средства реализации**

В данной работе используется язык программирования Python, за высокую скорость выполнения программ и широкий выбор библиотек.[3] Проект выполнен в среде разработки Visual Studio Code.

### 3.3 Листинг кода

На листингах 1, 2, 3 представлена реализация алгоритмов поиска в словаре.

```
1 def full_iteration(emails, val):
2     result = None
3     if len(emails) == 0:
4         return result
5     for key, value in emails.items():
6         if value == val:
7             result = key
8     return result
9
```

Листинг 1: Алгоритм полного перебора

```
1 def bin_search(emails, val):
2     result = None
3     if len(emails) == 0:
4         return result
5     emails = sort(emails, False)
6     mid, low, high = len(emails) // 2, 0, len(emails) - 1
7     tmp = tuple(emails.items())
8     while tmp[mid][1] != val and low <= high:
9         if val > tmp[mid][1]:
10             low = mid + 1
11         else:
12             high = mid - 1
13         mid = (low + high) // 2
14     if low < high:
15         result = tmp[mid][0]
16     return result
```

Листинг 2: Алгоритм бинарного поиска

```

1 def combined(emails, val):
2     result = None
3     if len(emails) == 0:
4         return result
5     arr = analyse(emails)
6     arr = sorted(arr, key = len, reverse = True)
7     copy = arr
8     for i in range(len(copy)):
9         tmp = tuple(copy[i].values())
10        if tmp[0][0] == val[0]:
11            result = bin_search(arr[i], val)
12    return result

```

Листинг 3: Алгоритм комбинированного поиска

### 3.4 Тестирование функций

В таблице 1 приведены функциональные тесты для функций, реализующих алгоритмы поиска в словаре. Все тесты прошли успешно.

Таблица 1: Результаты тестирования.

Ключ	Пустой словарь	Ожидаемый результат	Результат
ann78@hotmail.com	Нет	700983	700983
ann78@hotmail.com	Да	None	None
None	Нет	None	None

### 3.5 Вывод

В данном разделе была представлена структура ПО и листинги кода программы, а также результаты проведенного тестирования.

## 4 Исследовательская часть

В данном разделе приведены примеры работы и анализ характеристик разработанного программного обеспечения.

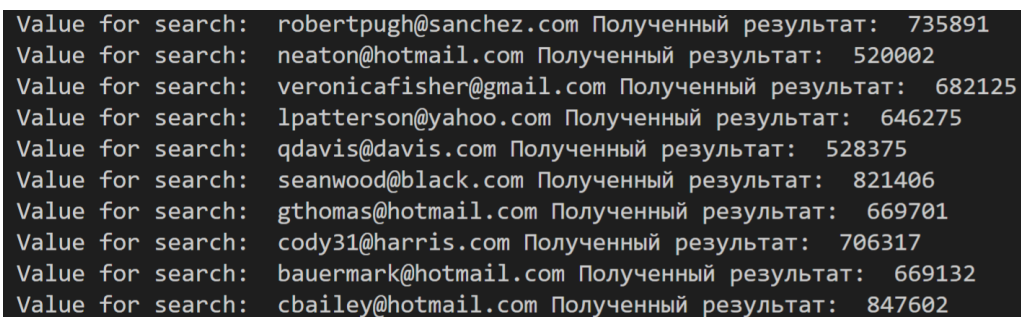
### 4.1 Системные характеристики

Характеристики компьютера на котором проводился замер времени сортировки массива:

1. операционная система - Windows 10;
2. процессор - Intel(R) Core(TM) i7-10510U CPU @1.80GHz 2.30GHz;
3. оперативная память - 16 ГБ;
4. количество ядер - 4;
5. количество логических процессов - 8.

### 4.2 Пример работы

Демонстрация работы программы приведена на рисунке 4.



```
Value for search: robertpugh@sanchez.com Полученный результат: 735891
Value for search: neaton@hotmail.com Полученный результат: 520002
Value for search: veronicafisher@gmail.com Полученный результат: 682125
Value for search: lpatterson@yahoo.com Полученный результат: 646275
Value for search: qdavis@davis.com Полученный результат: 528375
Value for search: seanwood@black.com Полученный результат: 821406
Value for search: gthomas@hotmail.com Полученный результат: 669701
Value for search: cody31@harris.com Полученный результат: 706317
Value for search: bauermark@hotmail.com Полученный результат: 669132
Value for search: cbailey@hotmail.com Полученный результат: 847602
```

Рис. 4: Пример работы программы.

### 4.3 Сравнительный анализ на основе замеров времени работы программы

Был проведен замер времени работы каждого алгоритма.

В таблице 2 показаны результаты эксперимента, суть которого заключается в анализе зависимости размера словаря от времени поиска при разных алгоритмах.

Таблица 2: Результаты эксперимента.

Размер словаря	Полный перебор	Бинарный поиск	Комбинированный
1000	0.00547088	0.00426798	5.42187500
2000	0.00661471	0.00486018	43.9218750
3000	0.00673328	0.00533766	293.583333
4000	0.00774536	0.00520833	845.364583
5000	0.00792178	0.01041666	2153.65624

### 4.4 Вывод

По проведенному анализу из эксперимента можно сделать вывод, что лучшее время поиска в словаре получается методом полного перебора, а худшее время - комбинированным способом. Комбинированный алгоритм показал такое время за счет анализа и сортировки значений в словаре. Так как во входном словаре данные могут быть уже отсортированы в обратном порядке, соответственно из-за этого получается такое плохое время. Бинарный алгоритм показал близкий результат с алгоритмом полного перебора. Но если оптимизировать сортировку в бинарном поиске и на вход дать уже отсортированный словарь, то он показал наилучшее время.

## Заключение

В ходе выполнения работы была достигнута цель выполнены все поставленные задачи:

1. рассмотреть и изучить алгоритмы полного перебора, двоичного поиска и эффективного поиска по словарю;
2. сравнить временные характеристики каждого из рассмотренных алгоритмов;
3. на основании проделанной работы сделать выводы.

Если исключить время сортировки из итогового времени, то алгоритмы двоичного поиска и частотного анализа должны показать похожие результаты. А за счет проведенного анализа, комбинированный метод показал бы наилучший результат.

## Список литературы

- [1] Александр. *Словари в Python 3 — основные методы и функции* [ЭЛ. РЕСУРС] Режим доступа: URL: <https://pythonru.com/osnovy/python-dict>. (дата обращения: 20.12.2020).
- [2] Лаборатория Линуксоида. *Двоичный поиск элемента* [ЭЛ. РЕСУРС] Режим доступа: URL: <https://younglinux.info/algorithm/dichotomy>. (дата обращения: 20.12.2020).
- [3] Гвидо ван Россум. *Python documentation* [ЭЛ. РЕСУРС] Режим доступа: URL: <https://docs.python.org/3/library/concurrent.futures.html>. (дата обращения: 22.11.2020).