Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

Лабораторная работа №7

по курсу «Анализ алгоритмов»

Поиск в словаре

Студент: Нгуен Фыок Санг

Группа ИУ7-56Б

Преподаватели: Волокова Л. Л., Строганов Ю.В.

Оценка:

Москва, 2020 г.

Оглавление

Введение

В данной лабораторной работе реализуются и оцениваются алгоритмы поиска элемента в массиве словарей по ключу, такие как алгоритм перебора, алгоритм бинарного поиска и сегментный поиск с частным анализом данных.

Поиск – это операция при котором производится обработка некоторого множества данных с целью выявления подмножества данных, которые соответствуют критериям поиска.

Словарь – абстрактный тип данных, который позволяет хранить пары вида "(ключ значение)"и поддерживающий операции добавления пары, а также поиска и удаления пары по ключу.

Поддержка словарей существует во многих интерпретируемых языках программирования высокого уровня, таких как Python, JavaScript, Ruby и других.

1. Аналитическая часть

Целью лабораторной работы является разработка и исследование алгоритмов поиска элемента в массиве словарей.

Можно выделить следующие задачи лабораторной работы:

- описание и реализация трёх алгоритмов поиска: алгоритм перебора, бинарного поиска и сегментного поиска с частным анализом данных;
- проведение замеров процессорного времени работы алгоритмов;
- анализ полученных результатов.

1.1. Поиск полным перебором

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. Двоичный поиск в упорядоченном словаре

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. Сегментный поиск с частным анализом

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 полученным 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, infor: string}. Поиск будет реализован по полю key.

1.5. Вывод

Результатом аналитического раздела стало определение цели и задач работы, описание используемых алгоритмов и описание самого словаря на котором будет проводится анализ.

2. Конструкторская часть

В данном разделе представлены схемы рассматриваемых алгоритмов и требования к программному обеспечению.

2.1. Схемы алгоритмов

На рисунках 2.1 - 2.3 представлены схемы вышеописанных алгоритмов поиска в словаре.

2.2. Требования к программному обеспечению

Для полноценного анализа и тестирования рассматриваемых алгоритмов необходимо обеспечить ряд требований:

- предоставить выбор алгоритма для поиска и обеспечить консольный ввод ключа, по которому будет происходить поиск;
- реализовать функцию замера процессорного времени, затраченного функциями.

2.3. Вывод

Результатом конструкторской части стало схематическое описание алгоритмов поиска в словаре и описание требований к программному обеспечению.

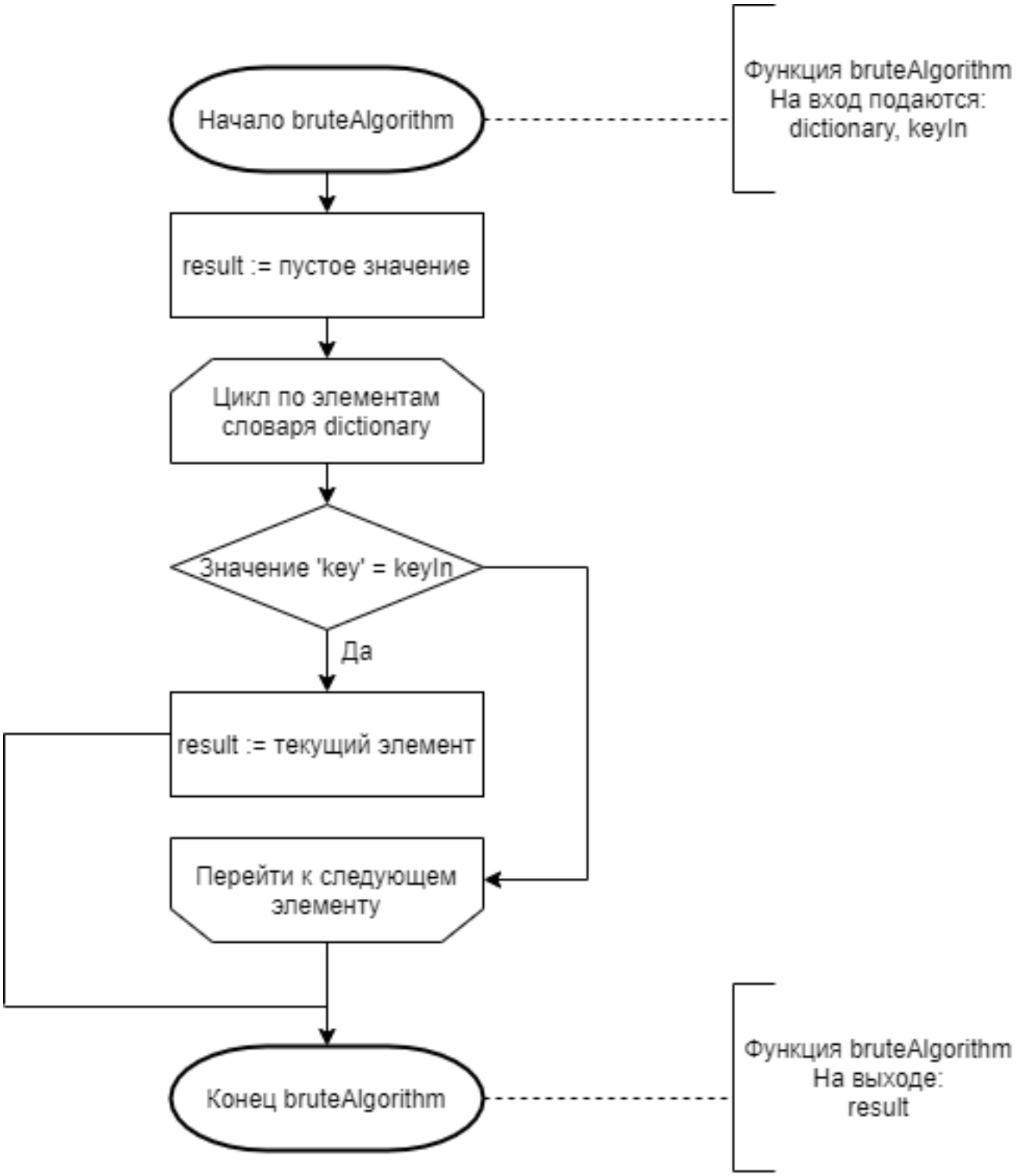

Рис. 2.1: схема алгоритма поиска перебором

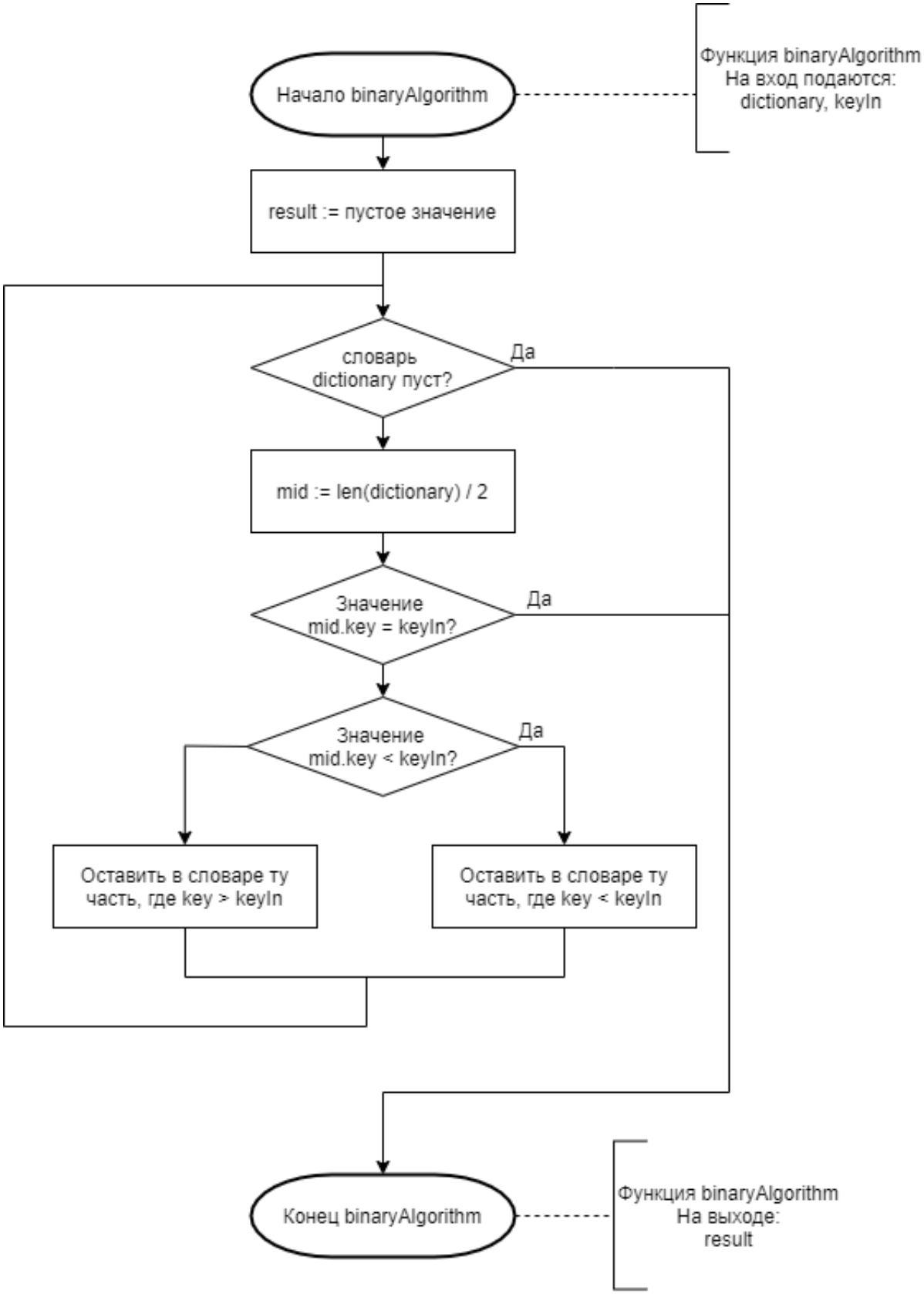

Рис. 2.2: схема алгоритма бинарного поиска

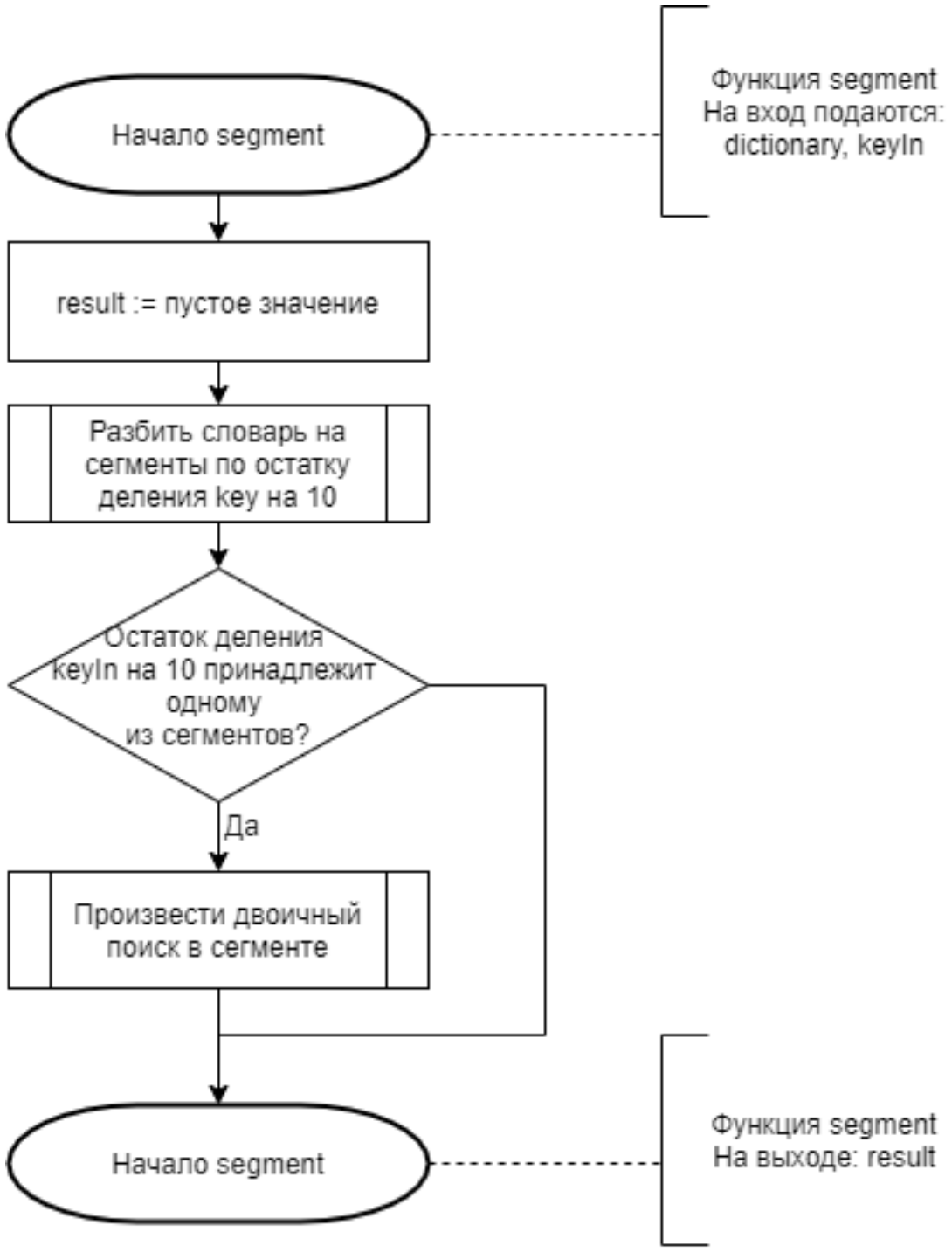

Рис. 2.3: схема алгоритма сегментного поиск с частным анализом

3. Технологическая часть

В данном разделе приведены средства для реализации рассматриваемых алгоритмов поиска, а также листинги кода.

3.1. Выбор языка программирования

В качестве языка программирования было решено выбрать Python 3, так как имеется опыт работы с библиотеками и инструментами языка, которые позволяют реализовать и провести исследования над алгоритмами поиска в словаре.

3.2. Реализация алгоритмов

В листингах 3.1 - 3.3 приведены реализации алгоритмов поиска в словаре.

Листинг 3.1: функция поиска перебором

```
1  def bruteAlgorithm(dictionary, key):
2      for item in dictionary:
3          if item['key'] == key:
4              return item
5
6      return None
```

Листинг 3.2: функция бинарного поиска

```
1  def binaryAlgorithm(dictionary, key):
2      Left = 0
3      Right = len(dictionary) - 1
4      mid = (Left + Right) // 2
5
6      if dictionary[Left]['key'] > key:
7          return None
8      elif dictionary[Left]['key'] == key:
9      return dictionary[Left]
10
11     if dictionary[Right]['key'] < key:
12         return None
13     elif dictionary[Right]['key'] == key:
14         return dictionary[Right]
15
16     tmp = dictionary[mid]['key']
17
18     while key != tmp and Left < Right:
19         if key < tmp:
```

```
20                Right = mid
21            else:
22                Left = mid
23
24            mid = (Left + Right) // 2
25            tmp = dictionary[mid]['key']
26
27        return dictionary[mid]
```

Листинг 3.3: функции для организации сегментного поиска

```
1  def func(x):
2      return x % 10
3
4  def prepareSegment(dictionary):
5      chance = [[] for i in range(10)]
6
7      for i in range(len(dictionary)):
8          chance[func(dictionary[i]['key'])].append(dictionary[i])
9
10     return chance
11
12
13 def segment(chance, key):
14     segment = chance[func(key)]
15     return binaryAlgorithm(segment, key)
```

3.3. Оценка времени

Для анализа времени алгоритмов поиска в словаре, реализована специальная функция, использующая библиотеку time [1]

3.4. Результаты тестирования

Все тесты прошли успешно.

3.5. Вывод

Результатом технологической части стал выбор используемых технических средств реализации и последующая реализация алгоритмов и замера времени работы на языке Python 3.

4. Исследовательская часть

Измерения процессорного времени проводятся на размерах словаря 100, 1000, 10000 и 100000. Содержание словаря сгенерировано случайным образом с помощью библиотеки faker.

Для повышения точности, каждый замер производится 100000 раз, за конечный результат берётся среднее арифметическое.

4.1. Результаты экспериментов

Эксперименты проводились на компьютере со следующими характеристиками:

- ОС - Windows 10, 64bit;
- Процессор - Intel Core i5 7200U 2.5GHz
- ОЗУ - 12Gb

По результатом измерений процессорного времени можно составить таблицу 4.1.

Таблица 4.1: результаты замеров процессорного времени работы алгоритмов

Название \ Размер	100	1000	10000	100000
Поиск перебором	$6.094 \cdot 10^{-6}$	$5.813 \cdot 10^{-5}$	$5 \cdot 10^{-4}$	$3.495 \cdot 10^{-3}$
Бинарный поиск	$4.688 \cdot 10^{-7}$	$7.813 \cdot 10^{-7}$	$1.250 \cdot 10^{-6}$	$6.718 \cdot 10^{-6}$
Сегментный поиск	$9.375 \cdot 10^{-7}$	$9.375 \cdot 10^{-7}$	$1.406 \cdot 10^{-6}$	$5.156 \cdot 10^{-6}$

4.2. Вывод

Анализируя полученные результаты, можно сделать вывод, что алгоритм поиска в словаре, использующий поиск по сегментам с частотным анализом, выигрывает по скорости у алгоритма полного перебора и бинарного поиска, а в ряде случаев проигрывает.

Связано это с тем, что для алгоритма сегментного поиска необходимо провести частотный анализ, а уже после проводить поиск по сегментам. В том случае, когда искомый элемент находится в начале словаря, алгоритм полного перебора будет выигрывать как у алгоритма двоичного поиска, так и у алгоритма частотного анализа.

По таблице видно, что бинарный поиск и поиск по сегментам имеют практически схожие результаты, но стоит учитывать, что в итоговое время не было включено время сортировки словаря для алгоритма бинарного поиска. Из этого можно заключить, что алгоритм поиска по сегментам будет работать быстрее.

Заключение

В ходе выполнения лабораторной работы достигнута поставленная цель: разработка и исследование алгоритмов поиска по словарю. Решены все задачи.

Были изучены и описаны понятия словаря и поиска по словарю. Также были описаны и реализованы поиск полным перебором, бинарный поиск и сегментный поиск с частотным анализом. Проведены замеры процессорного времени работы алгоритмов при различных размерах словаря. На основе экспериментов проведён сравнительный анализ.

Из проведённых экспериментов было выявлено, что бинарный поиск и поиск по сегментам имеют похожие результаты, но стоит учитывать, что в итоговое время не было включено время сортировки словаря для алгоритма бинарного поиска, поэтому можно предположить, что алгоритм поиска по сегментам с частотным анализом будет работать быстрее.

Список литературы

1. Документация на официальном сайте Python про библиотеку time, faker [Электронный ресурс].