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 Андрич К.

Группа ИУ7И-56Б

Оценка (баллы) _____

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Оглавление

Введение

Обычные списки (массивы) представляют собой набор пронумерованных элементов, то есть для обращения к какому-либо элементу списка необходимо указать его номер. Номер элемента в списке однозначно идентифицирует сам элемент. Но идентифицировать данные по числовым номерам не всегда оказывается удобно.

Структура данных, позволяющая идентифицировать ее элементы не по числовому индексу, а по произвольному, называется словарем или ассоциативным массивом. Каждый элемент словаря состоит из двух объектов: ключа и значения. В жизни широко распространены словари, например, привычные бумажные словари (толковые, орфографические, лингвистические). В них ключом является слово-заголовок статьи, а значением — сама статья. Для того, чтобы получить доступ к статье, необходимо указать слово-ключ.

Другой пример словаря, как структуры данных — телефонный справочник. В нем ключом является имя, а значением — номер телефона. И словарь, и телефонный справочник хранятся так, что легко найти элемент словаря по известному ключу (например, если записи хранятся в алфавитном порядке ключей, то легко можно найти известный ключ, например, бинарным поиском), но если ключ неизвестен, а известно лишь значение, то поиск элемента с данным значением может потребовать последовательного просмотра всех элементов словаря.

Особенностью ассоциативного массива является его динамичность: в него можно добавлять новые элементы с произвольными ключами и удалять уже существующие элементы. При этом размер используемой памяти пропорционален размеру ассоциативного массива. Доступ к элементам ассоциативного массива выполняется хоть и медленнее, чем к обычным массивам, но в целом довольно быстро.

Словари нужно использовать в следующих случаях:

1. Подсчет числа каких-то объектов. В этом случае нужно завести словарь, в котором ключами являются объекты, а значениями — их количество;

2. Хранение каких-либо данных, связанных с объектом. Ключи — объекты, значения — связанные с ними данные;

3. Установка соответствия между объектами (например, “родитель—потомок”). Ключ — объект, значение — соответствующий ему объект;

4. Если нужен обычный массив, но при этом максимальное значение индекса элемента очень велико, но при этом будут использоваться не все возможные индексы (так называемый “разреженный массив”), то можно использовать ассоциативный массив для экономии памяти.

Цель данной лабораторной работы: изучить и реализовать алгоритмы поиска по словарю. Для достижения поставленной цели требуется выполнить следующие задачи:

1. Реализовать алгоритм поиска по словарю с использованием полного перебора, двоичного поиска, комбинированного алгоритма;
2. Изучить алгоритм частотного анализа в задаче поиска по словарю;
3. Провести тестирование работы алгоритмов в худшем, лучшем и произвольном случаях;
4. Провести замеры процессорного времени работы алгоритмов поиска по словарю для каждого ключа и для отсутствующего ключа, вывести минимальное, максимальное и среднее время поиска ключа;
5. Сделать выводы на основе произведённого исследования и описать в отчёте.

1 | Аналитическая часть

В данном разделе будет поставлена цель и описаны задачи, описаны алгоритмы которые будут использоваться в лабораторной работе.

1.1 Алгоритм поиска в словаре с использованием полного перебора

Все элементы словаря пересматриваются последовательно в порядке их размещения, пока не найдётся элемент, равный ключу поиска. Если не известно, есть ли искомый элемент, то необходимо следить, чтобы поиск не вышел за границы списка. В таком случае техничным способом повышения эффективности программирования является введение стопера.

1.2 Алгоритм поиска в словре с использованием бинарного поиска

Бинарный поиск производится в упорядоченном словаре.

При бинарном поиске искомый ключ сравнивается с ключом среднего элемента в словаре. Если они равны, то поиск успешен. В противном случае поиск осуществляется аналогично в левой или правой частях словаря.

Алгоритм может быть определен в рекурсивной и нерекурсивной формах.

Бинарный поиск также называют поиском методом деления отрезка пополам или дихотомии.

На каждом шаге осуществляется поиск середины отрезка по формуле $mid = (left + right)/2$.

Если искомый элемент равен элементу с индексом mid, поиск завершается. В случае если искомый элемент меньше элемента с индексом mid, на место mid перемещается правая граница рассматриваемого отрезка, в противном случае — левая граница.

1.3 Комбинированный алгоритм

Перед началом работы алгоритма поиска для словаря, полученного на вход, нужно произвести сегментацию по первой букве, затем выполнить частотный анализ. Для этого нужно для первой буквы каждого значения в словаре посчитать, сколько раз она встречается в качестве первой буквы у других значений. По полученным значениям словарь разбивается на сегменты так, что все элементы с одинаковой первой буквой оказались в одном сегменте.

Сегменты сортируются по частоте первых букв. Далее каждый сегмент сортируется в лексикографическом порядке, чтобы можно было применить бинарный поиск в сегменте. Таким образом, сначала для искомого слова выбирается нужный сегмент, а затем в нем выполняется бинарный поиск

1.4 Вывод

В данном разделе поставлена цель и описаны задачи, описаны алгоритмы, которые будут использоваться в лабораторной работе.

2 | Конструкторская часть

В данном разделе будет представлено описание требований и допущений к программе и схемы алгоритмов.

2.1 Схемы алгоритмов

На рисунках 4.11, 2.2, 2.3, 2.4, 2.5 представлены схемы используемых алгоритмов.

На рисунке 4.11 представлен алгоритм полного перебора ключей.

На рисунке 2.2 представлен алгоритм бинарного поиска. На вход этот алгоритм получает отсортрованный в лексикографическом порядке словарь.

На рисунках 2.3 и 2.4 представлен алгоритм частотного анализа. В результате работы алгоритма словарь разбивается на сегменты и запысывается в список словарей. Каждый словарь в списке содержит поле с идентификатором сегмента (первый символ ключа), количеством слов в этом сегменте и поле «словарь», которое содержит элементы этого сегмента, отсортированные в лексикографическом порядке.

На рисунке 2.5 представлен комбинированный алгоритм, который получает на вход результат работы частноного анализа, выбирает подходящий сегмент и ищет в нём ключ бинарным поиском. Если подходящий сегмент не был найден, то ключа точно нет в словаре и поиск на этом можно прекратить.

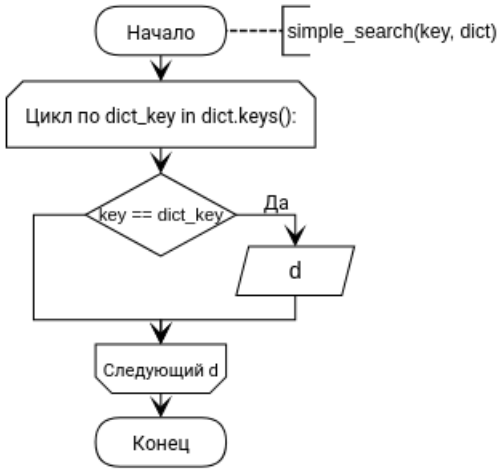

Рис. 2.1: Алгоритм полного перебора.

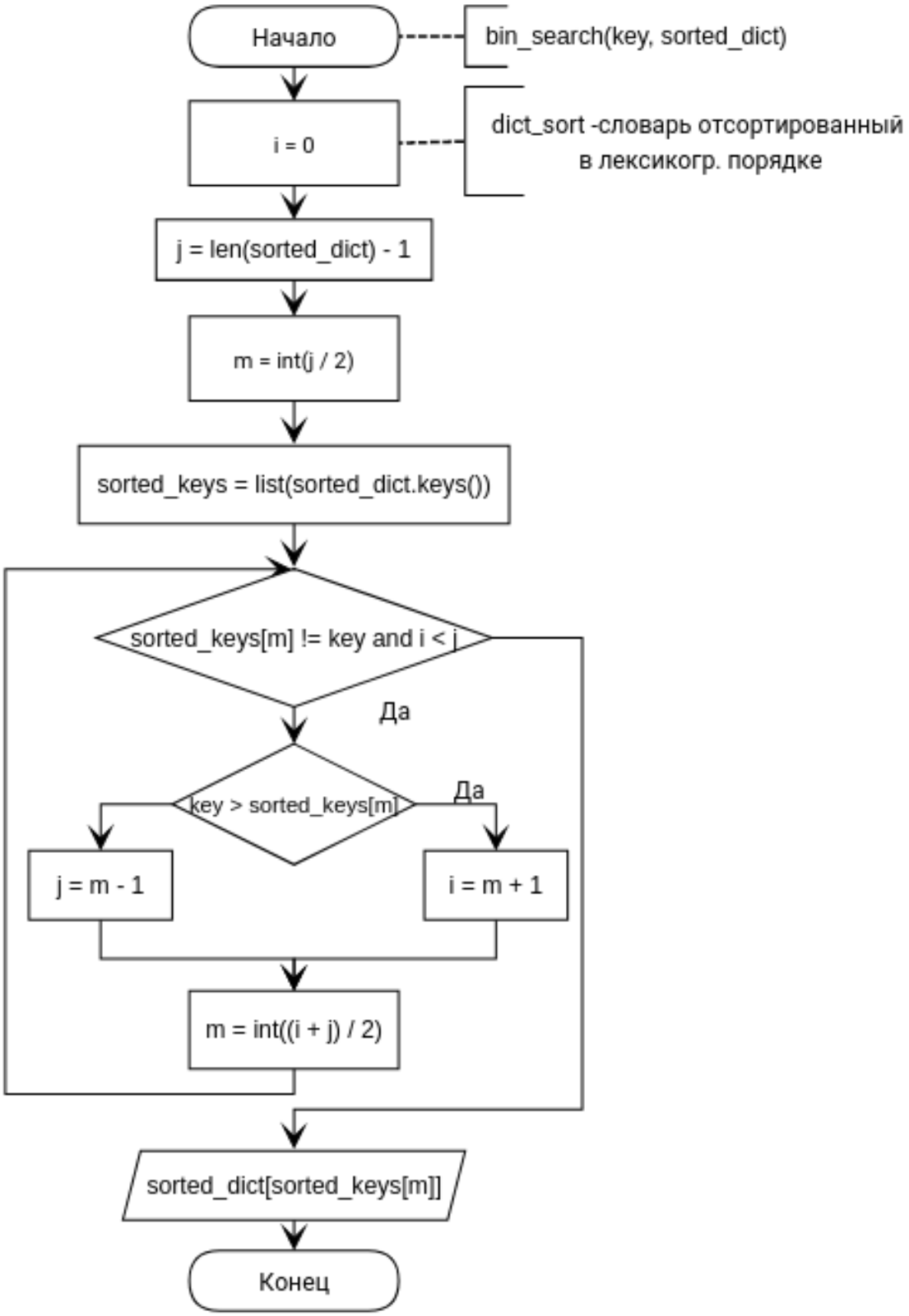

Рис. 2.2: Алгоритм бинарного поиска.

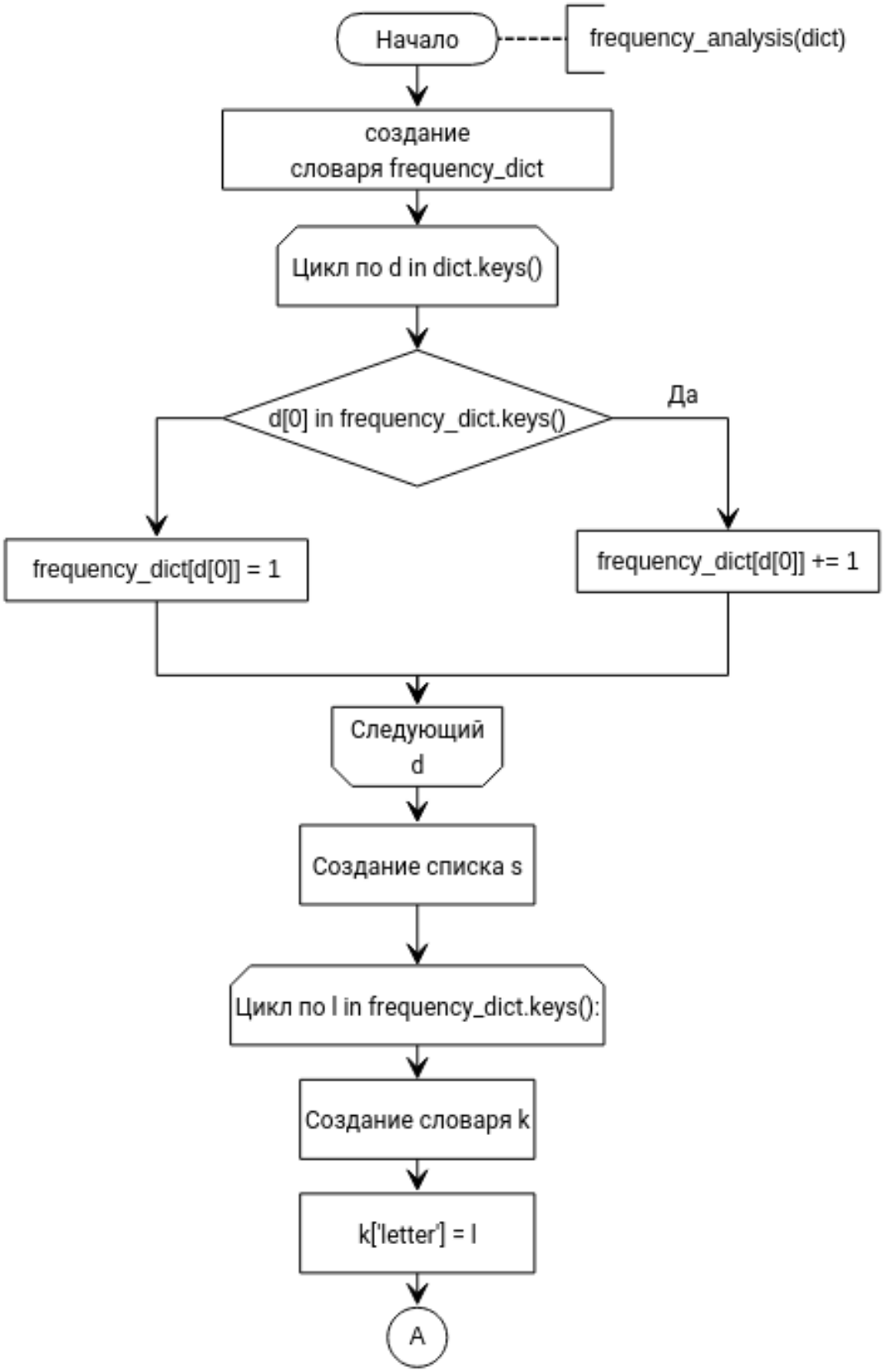

Рис. 2.3: Алгоритм частотного анализа, часть 1.

Рис. 2.4: Алгоритм частотного анализа, часть 2.

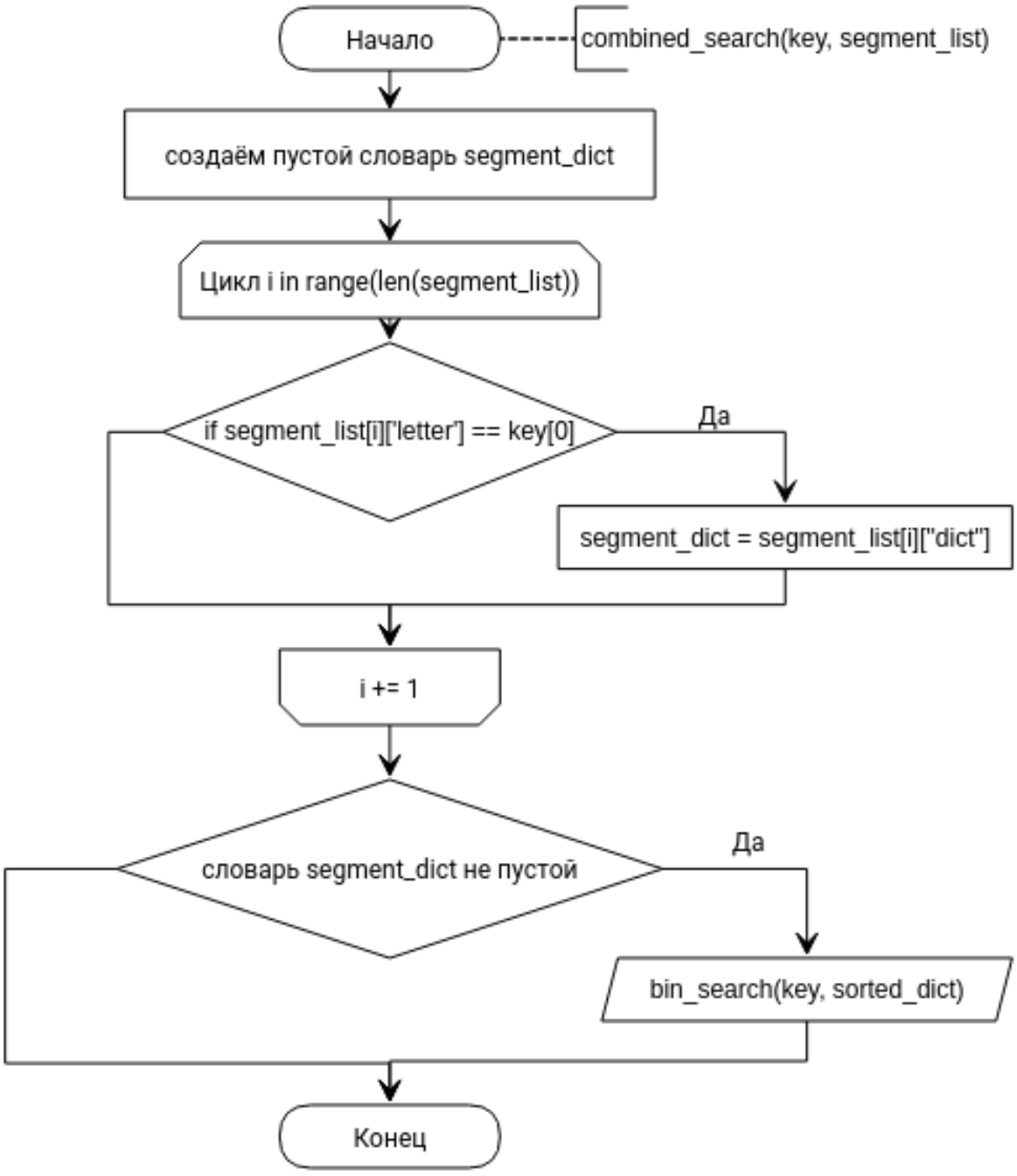

Рис. 2.5: Комбинированный алгоритм.

2.2 Требования к программному обеспечению

1. В словаре должно быть более 1000 вхождений.
2. Словарь является уникальным в пределах потока.

2.3 Вывод

В данном разделе представлено описание требований и допущений к программе и схемы алгоритмов.

3 | Технологическая часть

В этом разделе будет обоснован выбор языка програмирования и будут приведены листинги кода реализованных алгоритмов.

3.1 Выбор ЯП

В качестве языка программирования мной был выбран Python, так как этот язык удобен для работы с файлами и словарями. Для замера времени выполнения использовалась функция *time*() из библиотеки time.

3.2 Реализация алгоритмов

В листингах 3.1 - 3.4 представлена реализация алгоритмов.

Листинг 3.1: Алгоритм полного перебора.

```
1  def search_simple(key, dict):
2  for dict_key in dict.keys():
3  if key == dict_key:
4  return dict[key]
5  return None
```

Листинг 3.2: Алгоритм бинарного поиска.

```
1  def bin_search(key, sorted_dict):
2  sorted_keys = list(sorted_dict.keys())
3  i = 0
4  j = len(sorted_dict) - 1
5  m = int(j / 2)
6  while sorted_keys[m] != key and i < j:
7  if key > sorted_keys[m]:
8  i = m + 1
9  else:
10 j = m - 1
11 m = int((i + j) / 2)
12 if i > j:
13 return None
14 else:
15 return sorted_dict[sorted_keys[m]]
```

Листинг 3.3: Частотный анализ.

```
def frequency_analysis(dict):
frequency_dict = {}
for d in dict.keys():
if d[0] in frequency_dict.keys():
frequency_dict[d[0]] += 1
else:
frequency_dict[d[0]] = 1

s = []
for l in frequency_dict.keys():
k = {}
k['letter'] = l
k['count'] = frequency_dict[l]
k['dict'] = {}
for d in dict.keys():
if d[0] == l:
k['dict'][d] = dict[d]

sorted_list = sorted(k['dict'].items())
sorted_dict = {}
for i in range(len(sorted_list)):
sorted_dict[sorted_list[i][0]] = sorted_list[i][1]
k['dict'] = sorted_dict

s.append(k)


s.sort(key=lambda val: val['count'], reverse=True)
return s
```

Листинг 3.4: Комбинированный алгоритм.

```
def combined_search(key, segment_list):

segment_dict = {}
for i in range(len(segment_list)):
if segment_list[i]['letter'] == key[0]:
segment_dict = segment_list[i]["dict"]

if len(segment_dict) == 0:
return None

return bin_search(key, sorted_dict)
```

3.3 Тестовые данные

В таблице 3.1 представлено тестирование программы.
Тест 1 - ключ отсутсвует в словаре.
Тест 2 - ключ является первым элементом словаря.
Тест 3 - ключ является последним элементом словаря.
Тест 4 - ключ является произвольным элементом словаря.

Таблица 3.1: Функциональные тесты

Ключ	**Перебор**	**Бинарный поиск**	**Комбинированный**
’2000’	Ключ не найден	Ключ не найден!	Ключ не найден
’1’	’sad’	’sad’	’sad’
’1500’	’pa’	’pa’	’pa’
’23’	’je’	’je’	’je’

Фактические результаты тестов совпали с ожидаемыми результатами.

3.4 Вывод

В этом разделе обоснован выбор языка програмирования, описаны технические характеристики,приведены листинги кода реализованных алгоритмов. Программа прошла тестирование и работает правильно.

4 Исследовательская часть

В этом разделе будет приведена демонстрация работы программы и будет исследование полученных результатов.

4.1 Пример работы

Демонстрация работы программы приведена на рисунках 4.1, 4.2. На вход подаётся ключ для поиска в словаре.

```
Введите ключ: 458
Результат работы алгоритма полного перебора:
su
Результат работы алгоритма бинарного поиска:
su
Результат работы комбинированного алгоритма:
su
```

Рис. 4.1: Введённый ключ существует.

```
Введите ключ: 2000
Результат работы алгоритма полного перебора:
Ключ не найден
Результат работы алгоритма бинарного поиска:
Ключ не найден
Результат работы комбинированного алгоритма:
Ключ не найден
```

Рис. 4.2: Введённый ключ не существует.

4.2 Сравнение времени работы

На рисунке 4.3 представлено сравнение времени работы алгоритмов поиска по словарю перебором, бинарным поиском и комбинированным алгоритмом в лучшем, худшем,случайном случаях и при отсутствии ключа. Для каждого алгоритма представлено максимальное, минимальное и среднее время выполнения. Для замера времени каждая операция выполнялась 100 раз, среднее время выполнения взято как результат.

Для алгоритма поиска перебором лучший случай - это поиск слов, которые находятся первыми в словаре, худший случай - поиск последних слов словаря.

```
Лексикографическая сортировка  0.00108 секунд
Частотный анализ и лексикографическая сортировка  0.001972 секунд

Лучший случай поиска ключа
Алгоритм полного перебора 2.7179718017578123e-07 секунд
Алгоритм бинарного поиска 1.287221908569336e-05 секунд
Комбинированный алгоритм 2.3519992828369142e-05 секунд

Худший случай поиска ключа
Алгоритм полного перебора 7.373571395874023e-05 секунд
Алгоритм бинарного поиска 4.8003196716308595e-05 секунд
Комбинированный алгоритм 2.673149108886719e-05 секунд

Поиск несуществующего ключа
Алгоритм полного перебора 4.336118698120117e-05 секунд
Алгоритм бинарного поиска 1.6965866088867187e-05 секунд
Комбинированный алгоритм 9.34600830078125e-07 секунд

Поиск случайного ключа
Алгоритм полного перебора 2.8128623962402345e-05 секунд
Алгоритм бинарного поиска 1.8036365509033204e-05 секунд
Комбинированный алгоритм 1.640796661376953e-05 секунд

Алгоритм перебором
Максимальное время выполнения =  0.0001481151580810547
Минимальное время выполнения =  2.8133392333984374e-07
Среднее время выполнения =  3.207051229776482e-05

Бинарный поиск
Максимальное время выполнения =  6.932735443115234e-05
Минимальное время выполнения =  1.0755062103271484e-05
Среднее время выполнения =  1.9452014968845912e-05

Комбинированный алгоритм
Максимальное время выполнения =  7.044792175292969e-05
Минимальное время выполнения =  1.2617111206054688e-05
Среднее время выполнения =  2.0590693115983815e-05
```

Рис. 4.3: Время работы алгоритмов в разных случаях

Для алгоритма с бинарным поиском лучший случай - это поиск слова, которое находится в середине отсортированного в лексиграфическом порядке словаря, худший случай - слово находится при последнем делении диапазона поиска пополам.

Для комбинированного алгоритма лучший случай - это поиск слова, которое начинается на букву, которая чаще всего встречается в качестве первой буквы слова и находится в середине отсортированного в лексиграфическом порядке массива слов, начинающихся с этой же буквы; худший случай - слово, которое начинается на букву, которая реже всего встречается в качестве первой буквы слова и находится при последнем делении диапазона поиска пополам (в этом случае время выбора нужного сегмента будет больше, но время поиска в этом коротком сегменте меньше).

На рисунке 4.4 представлен график зависимости времени поиска от индекса ключа словаря, для построения которого использовались данные о времени поиска каждого элемента в словаре(1751 элемент). Индекс ключа указан на горизонтальной оси.

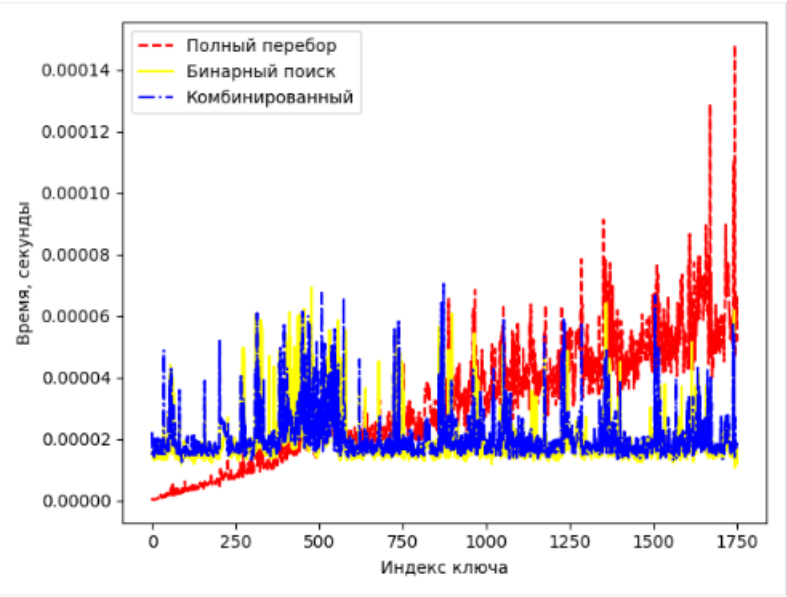

Рис. 4.4: Время работы алгоритмов для поиска каждого ключа словаря

Для большей наглядности будем использовать для постоения графика меньше точек(возьмём данные о времени поиска каждого пятнадцатого элемента словаря). График зависимости времени поиска от индекса ключа словаря представлен на рисунке 4.5.

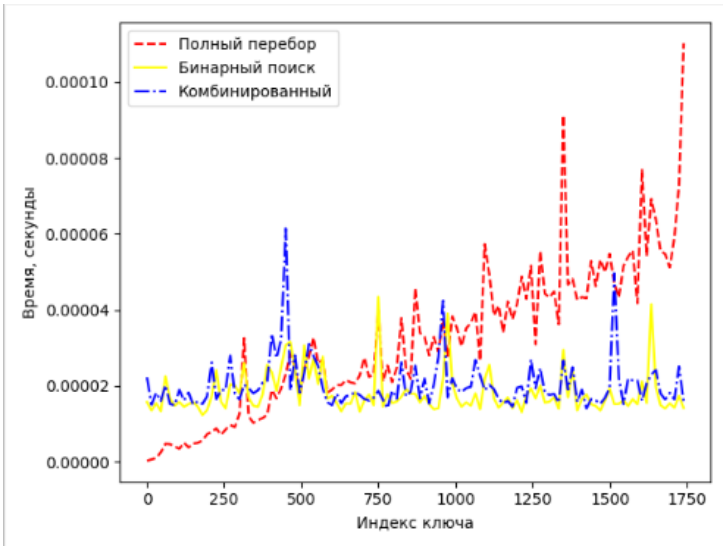

Рис. 4.5: Время работы алгоритмов для поиска каждого пятнадцатого ключа словаря

4.3 Гистограммы

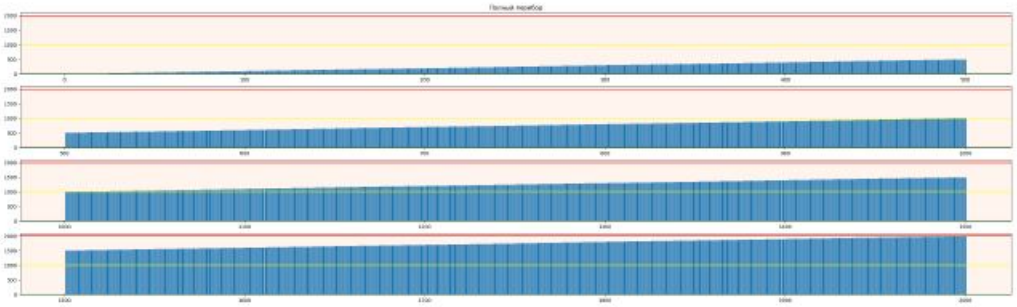

Рис. 4.6: Полный перебор, гистограмма типа 1

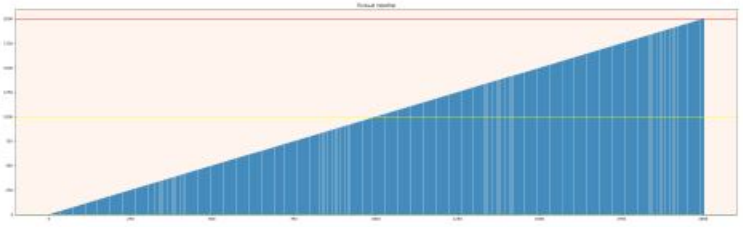

Рис. 4.7: Полный перебор, гистограмма типа 2

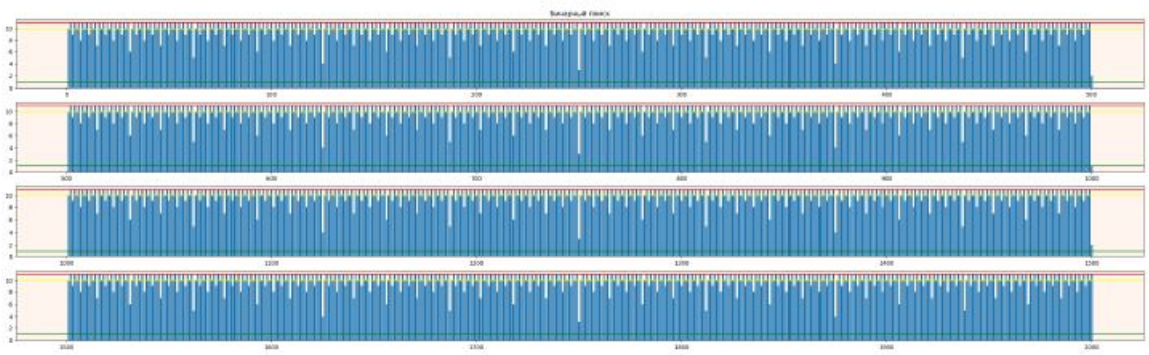

Рис. 4.8: Бинарный поиск, гистограмма типа 1

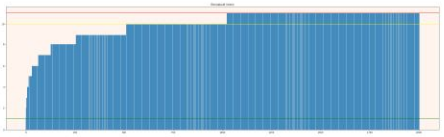

Рис. 4.9: Бинарный поиск, гистограмма типа 2

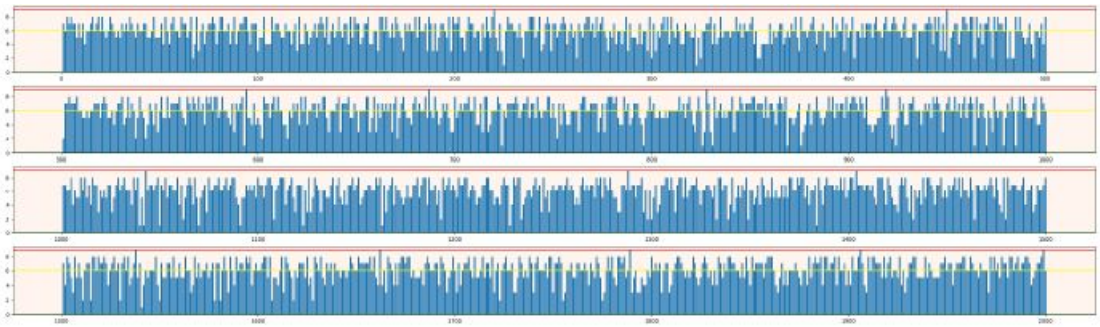

Рис. 4.10: Комбинированный алгоритм, гистограмма типа 1

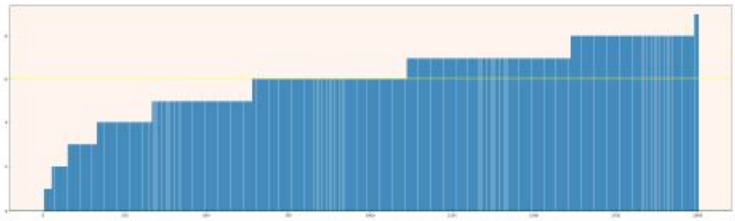

Рис. 4.11: Комбинированный алгоритм, гистограмма типа 2

4.4 Вывод

Как видно из графиков на рисунках 4.4, 4.5 самый медленный алгоритм - алгоритм полного перебора. Время в нём растет линейно и увеличивается с увеличением индекса элемента словаря. Алгоритм бинарного поиска и комбинированный алгоритм трятят примерно одинаковое количество времени, при этом они требуют дополнительных расходов времени на подготовку данных к работе с алгоритмом, но эти расходы малы.

Однако в лучшем случае алгоритм перебором показывает самые лучшие результаты. Он работает в 4 раза быстрее алгоритма бинарного поиска и в 8 раз быстрее комбинированного алгоритма.

В худшем случае алгоритм перебором затрачивает в 1.75 раза больше времени, чем бинарный поиск и в 3.5 раза больше времени чем комбинированный алгоритм. Комбинированный алгоритм в худшем случае работает быстрее остальных алгоритмов. При поиске несуществующего ключа комбинированный алгоритм тоже работает быстрее всех.

Заключение

В ходе данной лабораторной работы была достигнута цель: изучены и реализованы алгоритмы поиска по словарю.

Для достижения поставленной цели выполнены следующие задачи.

1. Реализован алгоритм поиска по словарю с использованием полного перебора, двоичного поиска, комбинированного алгоритма.
2. Изучен алгоритм частотного анализа в задаче поиска по словарю;
3. Проведено тестирование работы алгоритмов в худшем, лучшем и произвольном случаях.
4. Проведены замеры процессорного времени работы алгоритмов поиска по словарю для каждого ключа и для отсутствующего ключа, вывести минимальное, максимальное и среднее время поиска ключа.
5. Сделаны выводы на основе произведённого исследования.

Алгоритм полного перебора работает хорошо в лучшем случае (когда искомые ключи находятся в самом начале словаря). Однако для работы лучше использовать алгоритм бинарного поиска или комбинированный алгоритм. Они работают примерно одинаково по времени, но комбинированный алгоритм выигрывает в случае несуществующего ключа или худшем случае. В среднем алгоритм полного перебора работает в два раза дольше чем остальные алгоритмы.

Литература

[1] Словари (ассоциативные массивы) в Python [Электронный ресурс]. - Режим доступа: https://foxford.ru/wiki/informatika/slovari-assotsiativnye-massivy-v-python Дата обращения 30.11.2020.

[2] Алгоритмы последовательного поиска [Электронный ресурс]. - Режим доступа: https://cyberpedia.su/9x4e62.html Дата обращения 30.11.2020

[3] Бинарный поиск [Электронный ресурс]. Режим доступа: https://prog-cpp.ru/search-binary/ Дата обращения 30.11.2020

[4] Алгоритмы программирования [Электронный ресурс]. - Режим доступа: https://otus.ru/nest/post/829/ Дата обращения 30.11.2020

[5] Модуль time в Python [Электронный ресурс]. - Режим доступа: https://pythonru.com/osnovy/modul-time-v-python Дата обращения 01.12.2020