



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»

Отчёт по лабораторной работе №7 по дисциплине "Анализ алгоритмов"

Тема Поиск в словаре

Студент Романов А.В.

Группа ИУ7-53Б

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

Оглавление

Введение	2
1 Аналитическая часть	3
1.1 Полный перебор	3
1.2 Двоичный поиск	3
1.3 Частотный анализ	3
1.4 Описание словаря	4
2 Конструкторская часть	5
2.1 Разработка алгоритмов	5
3 Технологическая часть	9
3.1 Требование к ПО	9
3.2 Средства реализации	9
3.3 Реализация алгоритмов	9
3.4 Тестовые данные	10
4 Исследовательская часть	12
4.1 Технические характеристики	12
4.2 Время выполнения алгоритмов	12
Заключение	13
Литература	13

Введение

Цель лабораторной работы

Целью данной лабораторной работы является изучение алгоритмов поиска в словаре.

Задачи лабораторной работы

В рамках выполнения работы необходимо решить следующие задачи:

- реализовать три алгоритма поиска в словаре;
- замерить и сравнить время выполнения алгоритмов;
- сделать выводы на основе проделанной работы.

1 | Аналитическая часть

В данном разделе представлены теоретические сведения о рассматриваемых алгоритмах.

1.1 Полный перебор

Алгоритм полного перебора заключается в проходе по словарю, до того момента, пока не будет найден искомый ключ. В рассматриваемом алгоритме возможно $N + 1$ случаев расположения ключа: ключ является i -ым элементом словаря либо его нет в словаре в принципе.

Лучший случай (трудоемкость $O(1)$): ключ расположен в самом начале словаря и найден за одно сравнение). Худший случай (трудоемкость $O(N)$): ключ расположен в самом конце словаря либо ключ не находится в словаре.

1.2 Двоичный поиск

Данный алгоритм подходит только для заранее упорядоченного словаря.

Процесс двоичного поиска можно описать следующим образом:

- получить значение находящееся в середине словаря и сравнить его с ключом;
- в случае, если ключ меньше данного значения, продолжить поиск в младшей части словаря, в обратном случае – в старшей части словаря;
- на новом интервале снова получить значение из середины этого интервала и сравнить с ключом.
- поиск продолжать до тех пор, пока не будет найден искомый ключ, или интервал поиска не окажется пустым.

Обход словаря данным алгоритм можно представить в виде дерева, поэтому трудоемкость в худшем случае составит $\log_2 N$. Можно сделать вывод, что алгоритм двоичного поиска работает быстрее чем алгоритм полного перебора, но при этом требует предварительной обработки данных (сортировки).

1.3 Частотный анализ

Данный алгоритм также требует предварительной обработки данных, а именно:

- упорядочить словарь;
- разбить словарь на сегменты.

Словарь разбивается на сегменты по какому-либо признаку и сортируется по частоте. Например, если ключ является строкой, то можно сделать разбиение по первой букве в ключе. Если ключ является целым числом, можно провести разбиение по остатку от деления ключа на некоторое число K .

После выполнения разбиения, нужно определить к какому сегменту относится искомый ключ и провести на этом сегменте двоичный поиск.

Таким образом, время поиска в словаре увеличивается (особенно для самых часто встречаемых ключей), но, при этом, так же как и алгоритм двоичного поиска, частотный анализ требует предварительной обработки данных.

1.4 Описание словаря

Ключом в рассматриваемом мною словаре является ID преподавателя, а значением количество часов, которые он тратит на проведение лабораторных работ. Оба значения являются целочисленными. Так как значение ключа является целым числом, деление на сегменты будет производиться посредством получением остатка от деления ключа на некоторое число K .

Вывод

В данном разделе были рассмотрены особенности алгоритмов поиска в словаре.

2 | Конструкторская часть

В данном разделе представлены схемы рассматриваемых алгоритмов.

2.1 Разработка алгоритмов

На рисунках 2.1 - 2.3 приведены схема алгоритмов поиска в словаре.

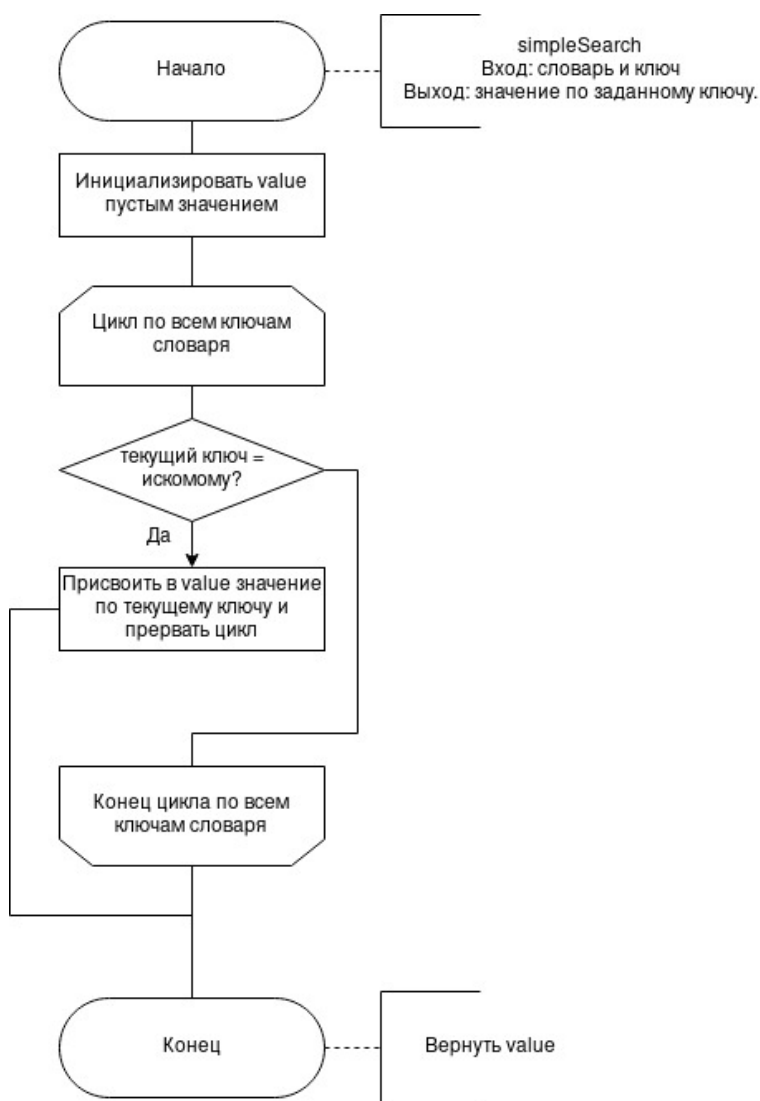


Рис. 2.1: Схема алгоритма полного перебора.

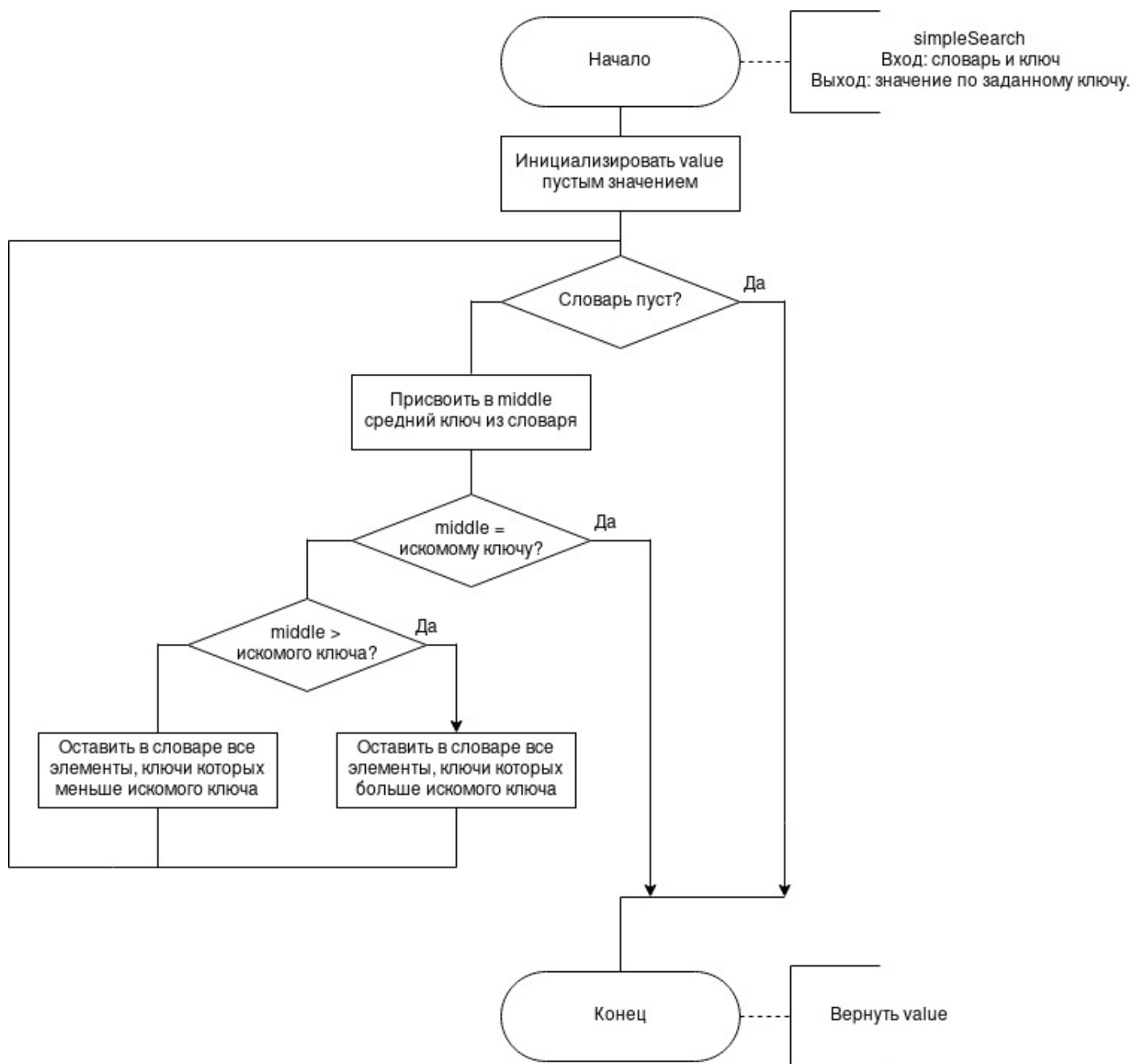


Рис. 2.2: Схема алгоритма двоичного поиска.

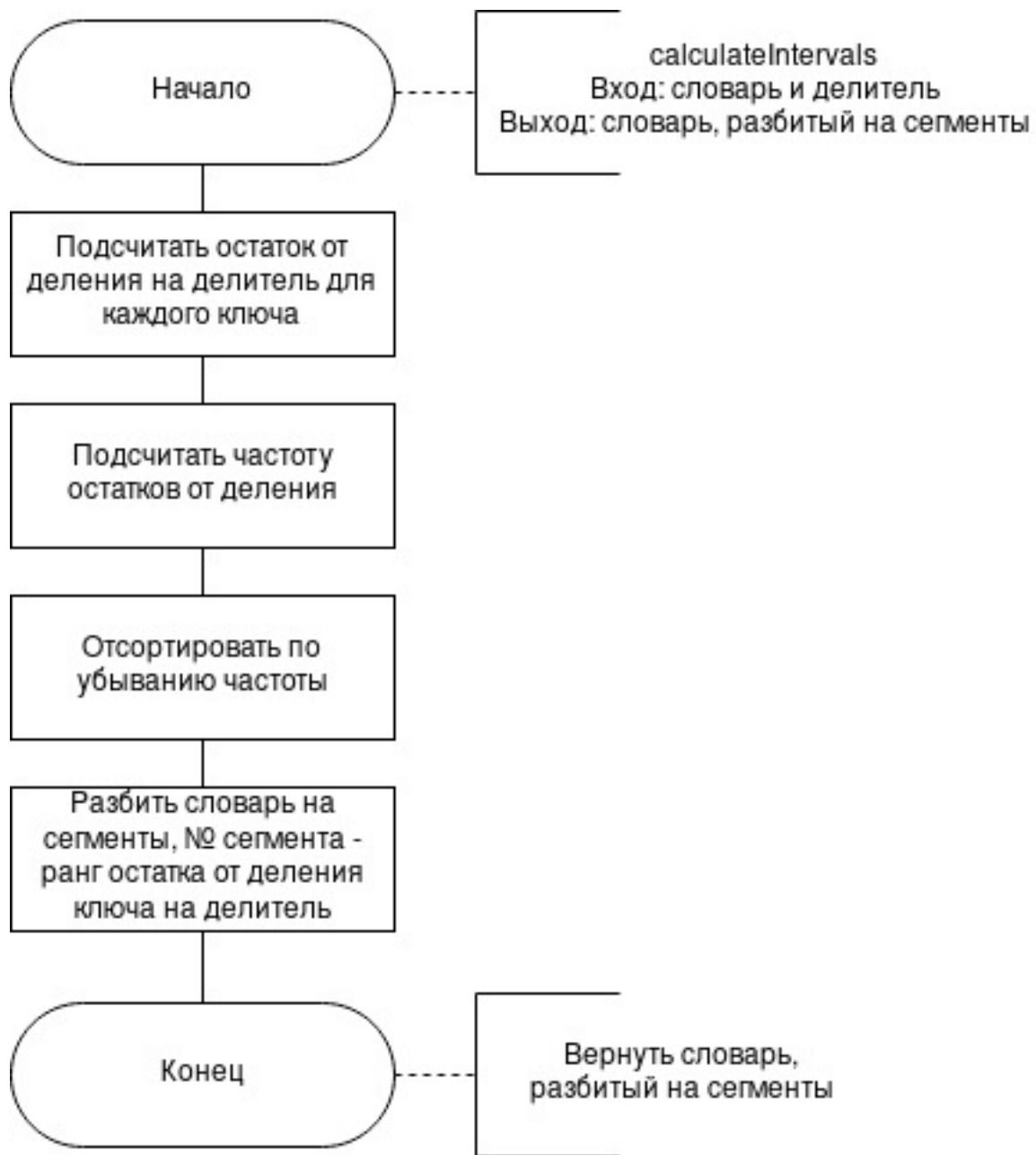


Рис. 2.3: Схема алгоритма частотного анализа.

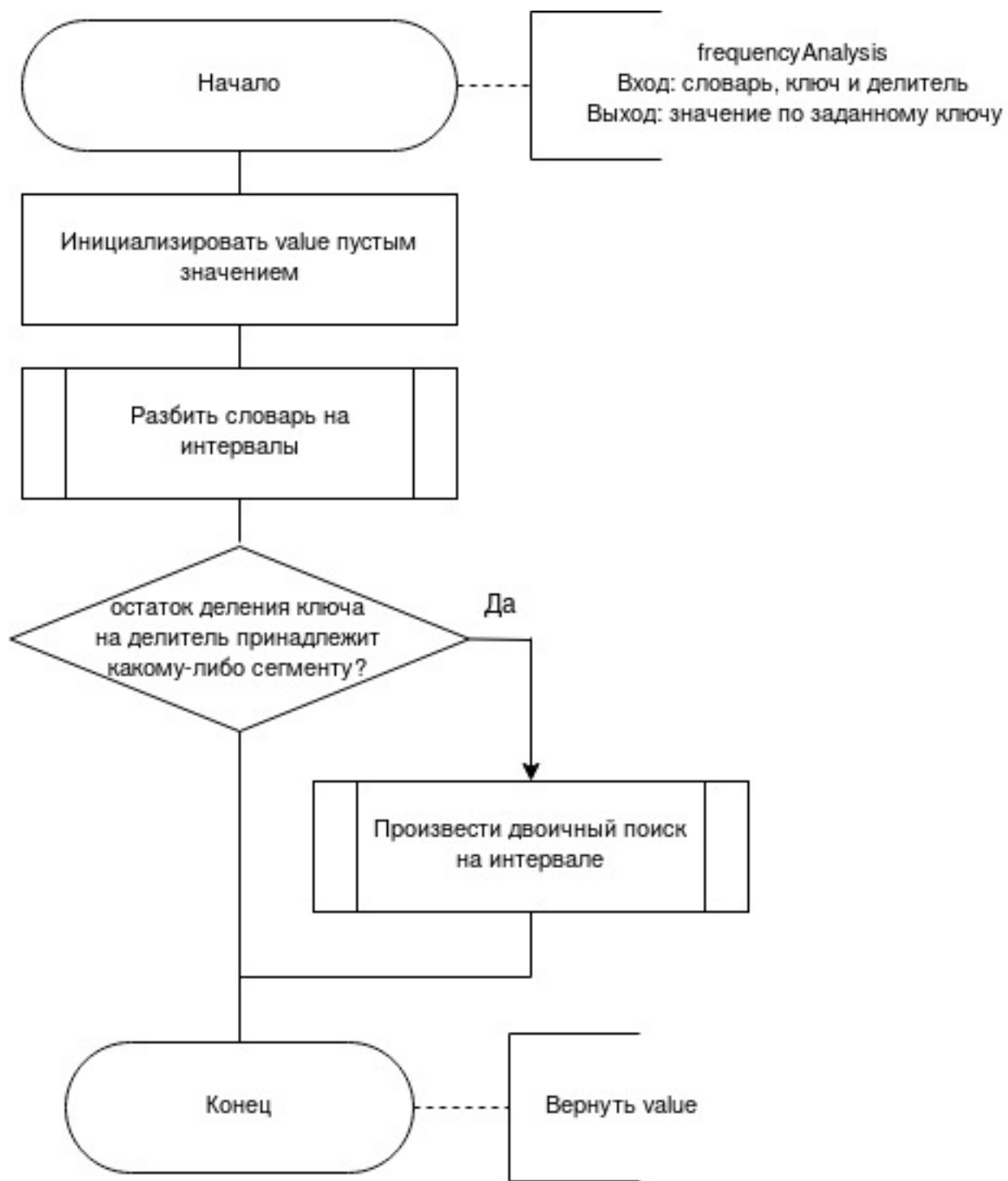


Рис. 2.4: Схема алгоритма поиска с использованием частотного анализа.

Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы алгоритмов поиска в словаре.

3 | Технологическая часть

В данном разделе приведены средства реализации и листинги кода.

3.1 Требование к ПО

К программе предъявляется ряд требований:

- на вход подается текстовый файл содержащий пары вида: ID преподавателя – количество часов потраченных на проведение лабораторных работ, кроме этого, с клавиатуры считывается искомый в словаре ключ (ID преподавателя);
- на выходе – значение для введенного с клавиатуры ключа.

3.2 Средства реализации

Для реализации ПО я выбрал язык программирования Haskell [1]. Данный выбор обусловлен моим желанием расширить свои знания в области применения данного языка программирования.

3.3 Реализация алгоритмов

В листингах 3.1 - 3.3 представлены листинги алгоритмов поиска в словаре.

Листинг 3.1: Алгоритм полного перебора

```
1 simpleSearch :: (Ord a, Eq b) => V.Vector (a, b) -> a -> Maybe b
2 simpleSearch dict key = simpleSearch' dict key $ V.head dict
3 where
4   simpleSearch' dict key curr | key == fst curr = Just $ snd curr
5   simpleSearch' dict key curr | V.null dict = Nothing
6   simpleSearch' dict key curr | otherwise = simpleSearch' (V.tail dict)
   key $ V.head dict
```

Листинг 3.2: Алгоритм двоичного поиска

```
1 binarySearch :: (Ord a, Eq b) => V.Vector (a, b) -> a -> Maybe b
2 binarySearch dict key = binarySearch' dict key $ middle dict
3 where
4   droppedSize dict = if V.length dict >= 2 then V.length dict else 2
```

```

5   fstHalf dict = V.take ((V.length dict) 'div' 2) dict
6   sndHalf dict = V.drop ((droppedSize dict) 'div' 2) dict
7   middle dict = dict V.! ((V.length dict) 'div' 2)
8
9   binarySearch' dict key curr | V.null dict = Nothing
10  binarySearch' dict key curr | key == fst curr = Just $ snd curr
11  binarySearch' dict key curr | key > fst curr = binarySearch' (sndHalf dict
    ) key (middle $ sndHalf dict)
12  binarySearch' dict key curr | key <= fst curr = binarySearch' (fstHalf
    dict) key $ (middle $ fstHalf dict)

```

Листинг 3.3: Алгоритм поиска с использованием частотного анализа

```

1 calculateIntervals :: Eq b => V.Vector (Int, b) -> Int -> V.Vector (V.Vector
    (Int, b))
2 calculateIntervals dict divider = intervals
3 where
4   remainder = V.foldl (\acc x -> acc V.++ (V.fromList [(fst x 'mod'
    divider, x)])) (V.empty) dict
5   intervals = V.foldl (
6     \acc x ->
7       acc V.// [(fst x, acc V.! (fst x) V.++ V.fromList [snd x])])
8     (V.generate divider (\x -> V.empty)) remainder
9
10
11 frequencyAnalysis :: Eq b => V.Vector (Int, b) -> Int -> Int -> Maybe b
12 frequencyAnalysis dict key divider =
13   if V.null $ disiredInterval key divider
14   then Nothing
15   else binarySearch (disiredInterval key divider) key
16   where
17     intervals = calculateIntervals dict divider
18     disiredInterval k d = intervals V.! (k 'mod' d)

```

3.4 Тестовые данные

В таблице 3.1 приведены тестовые данные, где ПП – полный перебор, ДП – двоичный поиск, ЧА – частотный анализ. Все тесты были пройдены успешно.

Таблица 3.1: Таблица тестовых данных алгоритмов поиска в словаре.

Входные данные	Ожидаемый результат	ПП	ДП	ЧА
1	217	217	217	217
145	338	338	338	338
876	111	111	111	111
1002	Нет ключа	Нет ключа	Нет ключа	Нет ключа

Вывод

В данном разделе была разработаны и протестированны алгоритмы поиска в словаре.

4 | Исследовательская часть

В данном разделе приведен анализ характеристик разработанного ПО.

4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено тестирование ПО:

- Операционная система: Debian [2] Linux [3] 11 «bullseye» 64-bit.
- Оперативная память: 12 GB.
- Процессор: Intel(R) Core(TM) i5-3550 CPU @ 3.30GHz [4].

4.2 Время выполнения алгоритмов

Время выполнения алгоритма замерялось с помощью применения технологии профайлинга [5]. Данный инструмент даёт детальное описание количество вызовов и количества времени CPU, затраченного на выполнение каждой функции.

В таблице 4.1 приведено сравнение времени выполнения алгоритмов: минимум, среднее, максимум.

Таблица 4.1: Таблица времени выполнения алгоритмов (в секундах)

Размер словаря	Полный перебор	Двоичный поиск	Частотный анализ
1000	X	X	X
10000	X	X	X
100000	X	X	X
1000000	X	X	X

Вывод

В данном разделе приведены время исполнения алгоритмов. Как видно из таблицы 4.1, вторая линия, то есть сборка двигателей (возведение числа в степень) занимает в среднем 70% времени от выполнения всей программы. Линия №3 в среднем работает быстрее чем линия №1.

Заключение

В рамках данной лабораторной работы лабораторной работы была достигнута её цель: изучены алгоритмы поиска в словаре. Также выполнены следующие задачи:

- реализованны три алгоритма поиска в словаре;
- замерено время выполнения алгоритмов;
- сделаны выводы на основе проделанной работы;

Что-то написать

Литература

- [1] Haskell Language. Режим доступа: <https://www.haskell.org/>. Дата обращения: 09.09.2020.
- [2] Debian – универсальная операционная система [Электронный ресурс]. Режим доступа: <https://www.debian.org/>. Дата обращения: 20.09.2020.
- [3] Linux – Getting Started [Электронный ресурс]. Режим доступа: <https://linux.org>. Дата обращения: 20.09.2020.
- [4] Процессор Intel® Core™ i5-3550 [Электронный ресурс]. Режим доступа: <https://ark.intel.com/content/www/ru/ru/ark/products/65516/intel-core-i5-3550-processor-6m-cache-up-to-3-70-ghz.html>. Дата обращения: 20.09.2020.
- [5] Glasgow Haskell Compiler – Profiling. Режим доступа: https://downloads.haskell.org/~ghc/latest/docs/html/users_guide/profiling.html. Дата обращения: 01.12.2020.