



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»
НАПРАВЛЕНИЕ ПОДГОТОВКИ _____ «09.03.04 Программная инженерия»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7

Тема: Поиск в словаре

Студент: Сироткина П.Ю.

Группа: ИУ7-56Б

Оценка: _____

Преподаватель: Волкова Л.Л.

Москва, 2021 г.

Содержание

Введение	3
1 Аналитический раздел	4
1.1 Постановка задачи	4
1.2 Алгоритм полного перебора	4
1.3 Алгоритм бинарного поиска	5
1.4 Алгоритм частотного анализа	5
1.5 Вывод	6
2 Конструкторский раздел	8
2.1 Схема алгоритма полного перебора	8
2.2 Схема алгоритма бинарного поиска	9
2.3 Схема алгоритма частотного анализа	10
2.4 Описание памяти, используемой программой	11
2.5 Структура ПО	11
2.6 Вывод	11
3 Технологический раздел	12
3.1 Средства реализации	12
3.2 Листинг кода	12
3.3 Тестирование ПО	15
3.4 Вывод	16
4 Экспериментальный раздел	17
4.1 Пример работы	17
4.2 Технические характеристики	17
4.3 Постановка эксперимента	18
4.4 Результаты эксперимента	18
4.5 Вывод	22
Заключение	23
Список использованных источников	24

Введение

Словарь, он же ассоциативный массив - абстрактный тип данных, позволяющий хранить пары вида (ключ;значение) и поддерживающий операции добавления, удаления и вставки. Также часто добавляются операции очищения словаря, поиска минимума, максимума и итерации через словарь. [1]

Очевидно, что для операций сравнения значений в словаре должно быть задано отношение порядка.

Словарь 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.

Поиск значения в словаре по ключу - основная операция для словарей.

Целью лабораторной работы является изучение алгоритмов поиска значения в словаре по ключу.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- Изучить алгоритм полного перебора, бинарного поиска и частотного анализа;
- Привести схемы изученных алгоритмов;
- Описать используемые типы данных;
- Описать структуру разрабатываемого ПО;
- Реализовать изученные алгоритмы;
- Протестировать разработанное ПО;
- Построить гистограммы 2 видов для каждого из алгоритмов: для гистограммы первого типа отсортировать ключи по алфавиту, а для второй - по количеству сравнений;
- Сделать выводы на основе проделанной работы.

1 Аналитический раздел

В данном разделе описаны основные идеи рассматриваемых алгоритмов. Поставлена задача, на примере которой будут исследоваться выбранные алгоритмы.[2]

1.1 Постановка задачи

В данной лабораторной работе поиск в словаре реализовывается на примере словаря, основанного на тексте сказок братьев Гримм. Ключом является номер слова в тексте, значением - соответственно само слово. Знаки препинания и прочие обозначения в тексте игнорируются.

1.2 Алгоритм полного перебора

Алгоритм полного перебора для любой задачи часто является самым примитивным и самым трудоемким алгоритмом, и в рамках рассматриваемой задачи этот случай не становится исключением.

Для поставленной задачи алгоритм полного перебора заключается в последовательном проходе по словарю до тех пор, пока не будет найден требуемый ключ. Очевидно, что худшим случаем является ситуация, когда необходимый ключ находится в конце словаря либо когда этот ключ вовсе не представлен в словаре.

Трудоемкость алгоритма зависит от положения ключа в словаре - чем дальше он от начала словаря, тем больше единиц процессорного времени потребуется на поиск.

Средняя трудоёмкость может быть рассчитана как математическое ожидание по формуле (1.1), где Ω – множество всех возможных случаев, k - кол-во единиц процессорного времени, затрачиваемого на одну операцию сравнения в словаре.

$$\begin{aligned}\sum_{i \in \Omega} p_i \cdot f_i &= k \cdot \frac{1}{N+1} + 2 \cdot k \cdot \frac{1}{N+1} + 3 \cdot k \cdot \frac{1}{N+1} + \dots + N \cdot k \cdot \frac{1}{N+1} = \\ &= \frac{k \cdot (1 + 2 + 3 + \dots + N)}{N+1} = \frac{k \cdot N \cdot (\frac{1+N}{2})}{N+1} = \frac{k \cdot N}{2}\end{aligned}\tag{1.1}$$

1.3 Алгоритм бинарного поиска

Алгоритм бинарного поиска применяется к заранее отсортированному набору значений. В рамках поставленной задачи словарь должен быть отсортирован по ключам по возрастанию.

Основная идея бинарного поиска для поставленной задачи заключается в следующем:

- Определение значения ключа в середине словаря. Полученное значение сравнивается с искомым ключом;
- Если ключ меньше значения середины, то поиск осуществляется в первой половине элементов, иначе — во второй;
- Поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с ключом;
- Процесс продолжается до тех пор, пока не будет найден элемент со значением ключа или не станет пустым интервал для поиска.

Как известно, сложность алгоритма бинарного поиска на заранее упорядоченном наборе данных составляет $O(\log_2(n))$, где n - размер словаря.

Однако, может возникнуть ситуация, что затраты на сортировку данных будут нивелировать преимущество быстрого поиска при больших размерностях массивов данных.

1.4 Алгоритм частотного анализа

Суть алгоритма частотного анализа, как видно из названия, заключается в разбиении исходного множества пар (ключ;значение) на некоторые т.н. сегменты по заранее заданному общему признаку.

Затем сегменты сортируются по их размеру, таким образом, сегмент, который содержит в себе больше всего вхождений, будет расположен первым, т.е. доступ к нему будет осуществляться быстрее, затем второй и так далее по убыванию.

Таким образом, на скорость доступа к сегменту влияет частота вхождений пар значений из словаря.

В каждом сегменте пары (ключ; значение) сортируются по возрастанию ключей для дальнейшего применения бинарного поиска в конкретном сегменте.

Таким образом, в начале алгоритм получает словарь и ключ для поиска. Сначала происходит обращение к наиболее частому сегменту, а в нем ищется совпадение ключа с помощью двоичного поиска. Если ключ не найден в первом сегменте, происходит обращение к следующему сегменту с последующим бинарным поиском до тех пор, пока не закончатся сегменты.

Если ключ не найден ни в одном сегменте, то ключ не представлен в данном словаре и возникает ошибка доступа.

Средняя трудоёмкость при множестве всех возможных случаев Ω может быть рассчитана по формуле (1.2).

$$\sum_{i \in \Omega} (f_{\text{выбор сегмента } i\text{-ого элемента}} + f_{\text{бинарный поиск } i\text{-ого элемента}}) \cdot p_i \quad (1.2)$$

1.5 Вывод

В данном разделе были изучены алгоритмы поиска в словаре: алгоритм полного перебора, алгоритм бинарного поиска, алгоритм частотного анализа.

Требования к ПО

К программе предъявляются следующие требования:

- Текст сказок братьев Гримм для наполнения словаря содержится в папке data в файле fairytale.txt;
- Массив слов из текста (без знаков препинания и прочих символов, не являющихся буквами) с содержится в папке data в файле data.txt;
- Программа не проверяет корректность указанных файлов;
- Программа должна предоставлять пользовательский интерфейс и корректно реагировать на любое действие пользователя;
- Принцип разбиения на сегменты - анализ первой буквы слова;

- Поиск в пустом словаре рассматривается как ошибочная ситуация;
- Программа должна производить логирование для каждого из рассмотренных алгоритмов - создаются файлы с соответствующим названием, содержащие строки вида (номер ключа; количество сравнений во время поиска).

2 Конструкторский раздел

В данном разделе представлены схемы муравьиного алгоритма и алгоритма полного перебора для задачи коммивояжера.

2.1 Схема алгоритма полного перебора

На рисунке 2.1 представлена схема алгоритма полного перебора для задачи поиска в словаре.

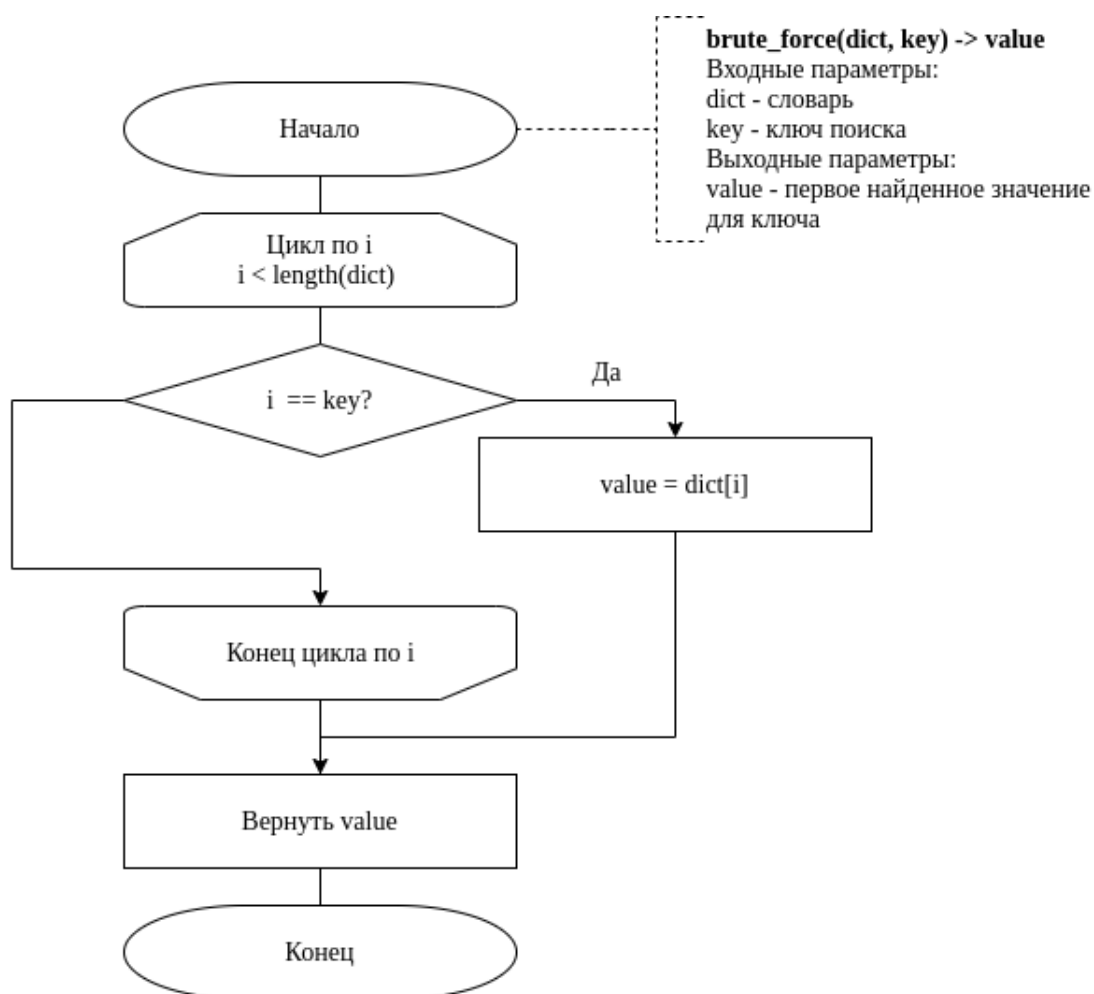


Рисунок 2.1 — Схема алгоритма полного перебора для задачи поиска в словаре

2.2 Схема алгоритма бинарного поиска

На рисунке 2.2 представлена схема алгоритма бинарного поиска для задачи поиска в словаре.

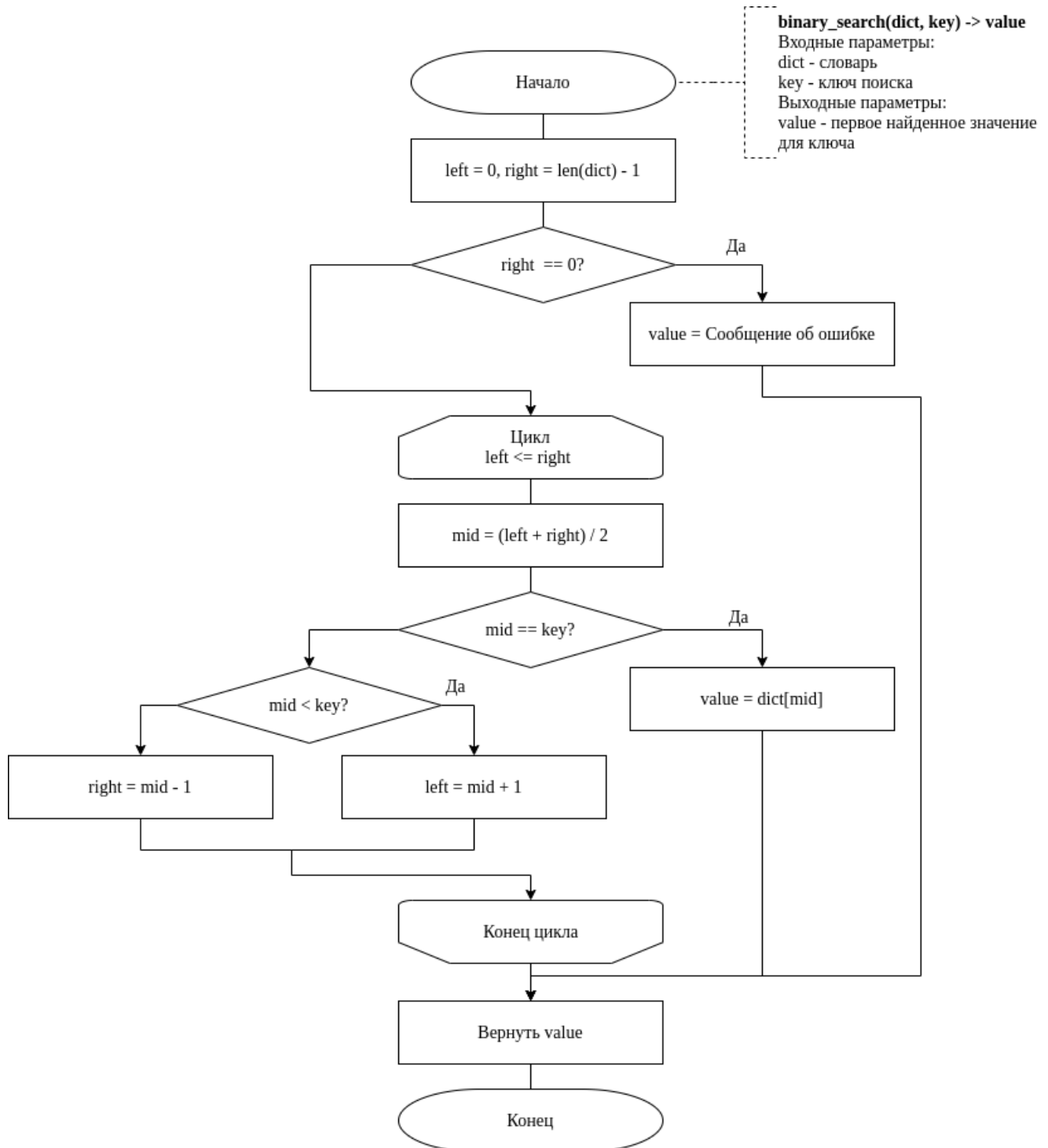


Рисунок 2.2 — Схема алгоритма бинарного поиска для задачи поиска в словаре

2.3 Схема алгоритма частотного анализа

На рисунке 2.3 представлена схема алгоритма частотного анализа для задачи поиска в словаре.

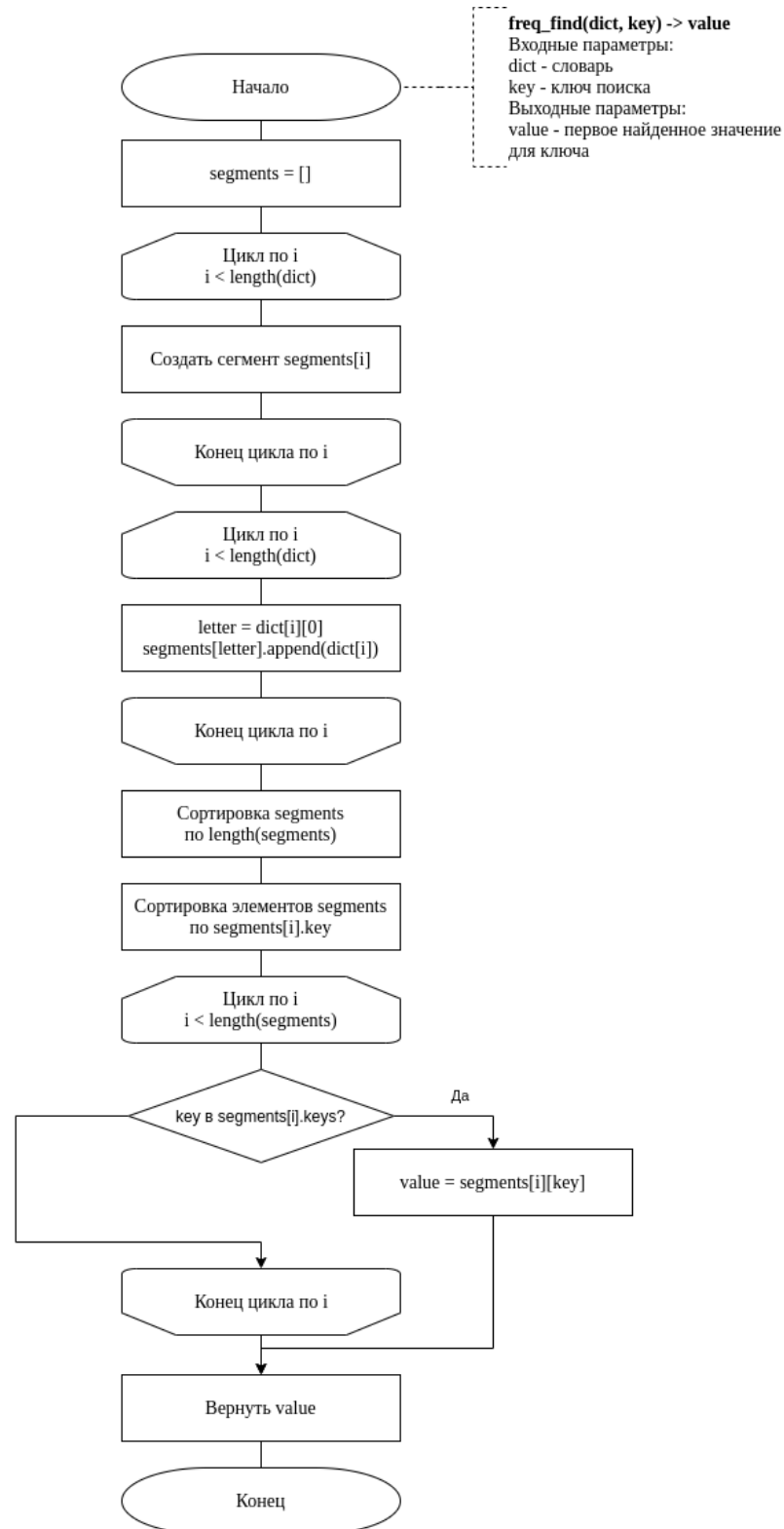


Рисунок 2.3 — Схема алгоритма частотного анализа для задачи поиска в словаре

2.4 Описание памяти, используемой программой

Затраты с точки зрения памяти на функционирование ПО складываются в большей степени за счет количества слов в словаре, типов ключей и значений, а также за счет поддержания пользовательских типов данных.

2.5 Структура ПО

Программа будет включать в себя один смысловой модуль, называемый **find**, который содержит в себе процедуры и функции, связанные с реализацией алгоритмов поиска в словаре по ключу. Независимо от модуля будет существовать файл `main.go`, реализующий мост между пользователем и модулем, описанным ранее. Также реализована программа на языке `python`, которая формирует данные для словаря из текста сказок.

2.6 Вывод

На основе теоретических данных, полученных из аналитического раздела, были разработаны схемы требуемых алгоритмов. Рассмотрены затраты по памяти и описана структура ПО.

3 Технологический раздел

В данном разделе приведены средства реализации и листинги кода.

3.1 Средства реализации

В качестве языка программирования для реализации лабораторной работы был выбран язык Go. Выбор этого языка обусловлен его быстрой работой и эффективностью, это лаконичный и гибкий язык.

3.2 Листинг кода

В листинге 3.1 представлены используемые типы данных для реализации алгоритмов поиска.

Листинг 3.1 — Реализация используемых типов данных

```
1 type Dict_t map[int]string
2
3 type Segment_t Dict_t
```

В листинге 3.2 представлена реализация алгоритма полного перебора для поставленной задачи.

Листинг 3.2 — Реализация алгоритма полного перебора

```
1 func BruteForce(dict Dict_t, key int) string {
2     for i := 0; i < len(dict); i++ {
3         if key - 1 == i {
4             return dict[i]
5         }
6     }
7     return "ERROR:NOT_FOUND"
8 }
```

В листинге 3.3 представлена реализация алгоритма бинарного поиска для поставленной задачи.

Листинг 3.3 — Реализация алгоритма бинарного поиска

```
1 func BinarySearch(dict Dict_t, key int) string {
2     var (left int
3         right int
4         mid int)
5 }
```

```

6     left = 0
7     right = len(dict) - 1
8
9     if right != 0 {
10        for left <= right {
11            mid = left + (right - left) / 2
12            if mid == key - 1 {
13                return dict[mid]
14            }
15            if mid < key - 1 {
16                left = mid + 1
17            } else {
18                right = mid - 1
19            }
20        }
21    }
22    return "ERROR:NOT_FOUND"
23 }

```

В листинге 3.4 представлена реализация алгоритма частотного анализа для поставленной задачи.

Листинг 3.4 — Реализация алгоритма частотного анализа

```

1  var alphabet = string("abcdefghijklmnopqrstuvwxyz")
2  var mask = make([]int, len(alphabet))
3
4  func FreqFind(dict Dict_t, key int) string {
5      init_mask()
6      segments := create_segments(len(alphabet))
7      divide_by_first_letter(segments, dict)
8      sort_by_size(segments)
9      return binary_search(segments, key - 1)
10 }
11
12 func init_mask() {
13     for i := 0; i < len(alphabet); i++ {
14         mask[i] = i
15     }
16 }
17
18 func create_segments(size int) []Segment_t {
19     segments := make([]Segment_t, size)
20     for i := 0; i < size; i++ {
21         segments[i] = make(Segment_t)
22     }
23     return segments

```

```

24 }
25
26 func divide_by_first_letter(segments []Segment_t, dict Dict_t) {
27     for i := 0; i < len(dict); i++ {
28         index := find_pos(strings.ToLower(string(dict[i][0])))
29         segments[index][i] = dict[i]
30     }
31 }
32
33 func sort_by_size(segments []Segment_t) {
34     for i := 0; i < len(segments) - 1; i++ {
35         for j := 0; j < len(segments) - i - 1; j++ {
36             if len(segments[j]) < len(segments[j + 1]) {
37                 swap_segs(segments, j, j + 1)
38                 swap_in_int_arr(mask, j, j + 1)
39             }
40         }
41     }
42 }
43
44 func binary_search(segments []Segment_t, key int) string {
45     var (left int
46         right int
47         mid int)
48
49     for i := 0; i < len(segments); i++ {
50         var keys = make([]int, 0)
51         for key := range(segments[i]) {
52             keys = append(keys, key)
53         }
54         sort_keys(keys)
55
56         left = 0
57         right = len(segments[i]) - 1
58
59         if right != 0 {
60             for left <= right {
61                 mid = left + (right - left) / 2
62                 if keys[mid] == key {
63                     return segments[i][keys[mid]]
64                 }
65                 if keys[mid] < key {
66                     left = mid + 1
67                 } else {
68                     right = mid - 1
69                 }

```

```

70         }
71     }
72 }
73     return "ERROR:NOT_FOUND"
74 }
75
76 func find_pos(a string) int {
77     pos := -1
78     for i := 0; i < len(alphabet) && pos == -1; i++ {
79         if string(alphabet[i]) == a {
80             pos = i
81         }
82     }
83     return pos
84 }
85
86 func swap_segs(s []Segment_t, i int, j int) {
87     temp := s[i]
88     s[i] = s[j]
89     s[j] = temp
90 }
91
92 func swap_in_int_arr(a []int, i int, j int) {
93     temp := a[i]
94     a[i] = a[j]
95     a[j] = temp
96 }
97
98 func sort_keys(keys []int) {
99     for i := 0; i < len(keys) - 1; i++ {
100         for j := 0; j < len(keys) - i - 1; j++ {
101             if keys[j] > keys[j + 1] {
102                 swap_in_int_arr(keys, j, j + 1)
103             }
104         }
105     }
106 }

```

3.3 Тестирование ПО

В таблице 3.1 приведены тестовые данные, на которых было протестировано разработанное ПО. Все тасты были успешно пройдены.

Таблица 3.1 — Тестирование алгоритмов

Исходный словарь	Ключ	Ожидание	ПП	БП	ЧА
1 : 'a', 2 : 'b', 3 : 'c'	1	'a'	'a'	'a'	'b'
1 : 'a', 2 : 'b', 3 : 'c'	2	'b'	'b'	'b'	'b'
1 : 'a', 2 : 'b', 3 : 'c'	3	'c'	'c'	'c'	'c'
1 : 'a', 2 : 'b', 3 : 'c'	-3	Ошибка	Ошибка	Ошибка	Ошибка
1 : 'a', 2 : 'b', 3 : 'c'	4	Ошибка	Ошибка	Ошибка	Ошибка
1 : 'a', 2 : 'b', 3 : 'c', 4: 'c'	3	'c'	'c'	'c'	'c'
1 : 'a', 2 : 'b', 3 : 'c', 2: 'b'	3	'b'	'b'	'b'	'b'
Пустой словарь	5	Ошибка	Ошибка	Ошибка	Ошибка
Пустой словарь	0	Ошибка	Ошибка	Ошибка	Ошибка

Обозначения:

- ПП - алгоритм полного перебора;
- БП - бинарный поиск;
- ЧА - частотный анализ.

3.4 Вывод

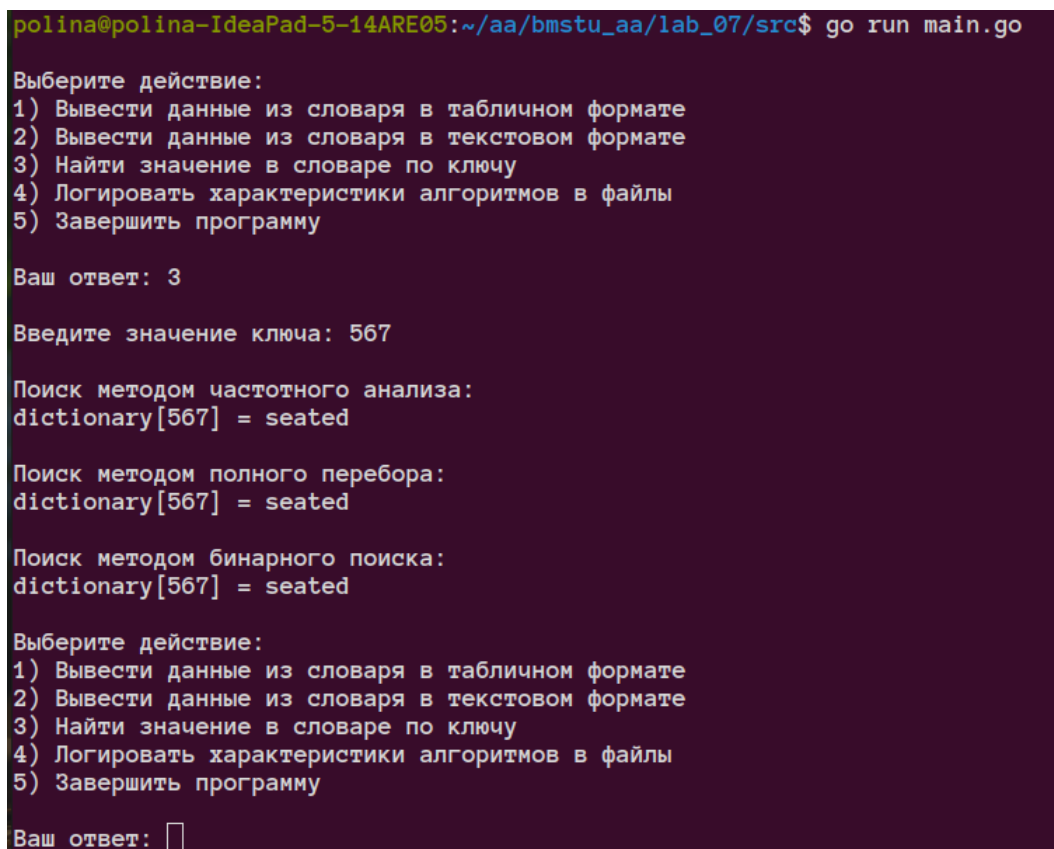
В данном разделе было представлено реализованное ПО для решения поставленной задачи, а также проведено тестирование.

4 Экспериментальный раздел

В данном разделе представлен пользовательский интерфейс, а также проведена оценка эффективности алгоритмов.

4.1 Пример работы

На рисунке 4.1 приведен пример работы программы.



```
polina@polina-IdeaPad-5-14ARE05:~/aa/bmstu_aa/lab_07/src$ go run main.go
Выберите действие:
1) Вывести данные из словаря в табличном формате
2) Вывести данные из словаря в текстовом формате
3) Найти значение в словаре по ключу
4) Логировать характеристики алгоритмов в файлы
5) Завершить программу

Ваш ответ: 3

Введите значение ключа: 567

Поиск методом частотного анализа:
dictionary[567] = seated

Поиск методом полного перебора:
dictionary[567] = seated

Поиск методом бинарного поиска:
dictionary[567] = seated

Выберите действие:
1) Вывести данные из словаря в табличном формате
2) Вывести данные из словаря в текстовом формате
3) Найти значение в словаре по ключу
4) Логировать характеристики алгоритмов в файлы
5) Завершить программу

Ваш ответ: 
```

Рисунок 4.1 — Пример работы программы

4.2 Технические характеристики

Технические характеристики машины, на которой выполнялось тестирование:

- Операционная система: Ubuntu[3] Linux[4] 20.04 64-bit;
- Оперативная память: 16 Gb;
- Процессор: AMD(R) Ryzen(TM)[5] 5 4500U CPU @ 2.3 CHz.

4.3 Постановка эксперимента

Эксперимент заключается в генерации гистограмм на основе ключей и соответствующих им количеств сравнений для поиска ключа.

Для каждого из алгоритмов строятся гистограммы 2 видов: для гистограммы первого типа отсортировать ключи по алфавиту, а для второй - по количеству сравнений.

По полученным гистограммам необходимо сделать выводы об эффективности алгоритмов.

Необходимо также отметить среднее, минимальное и максимальное количество сравнений.

4.4 Результаты эксперимента

На рисунках 4.2 - 4.7 представлены результаты эксперимента.

По оси X откладываются значения ключей, для оси Y - количество сравнений.

Зеленой горизонтальной линией отмечено минимальное количество сравнений, желтой - среднее, красной - максимальное.

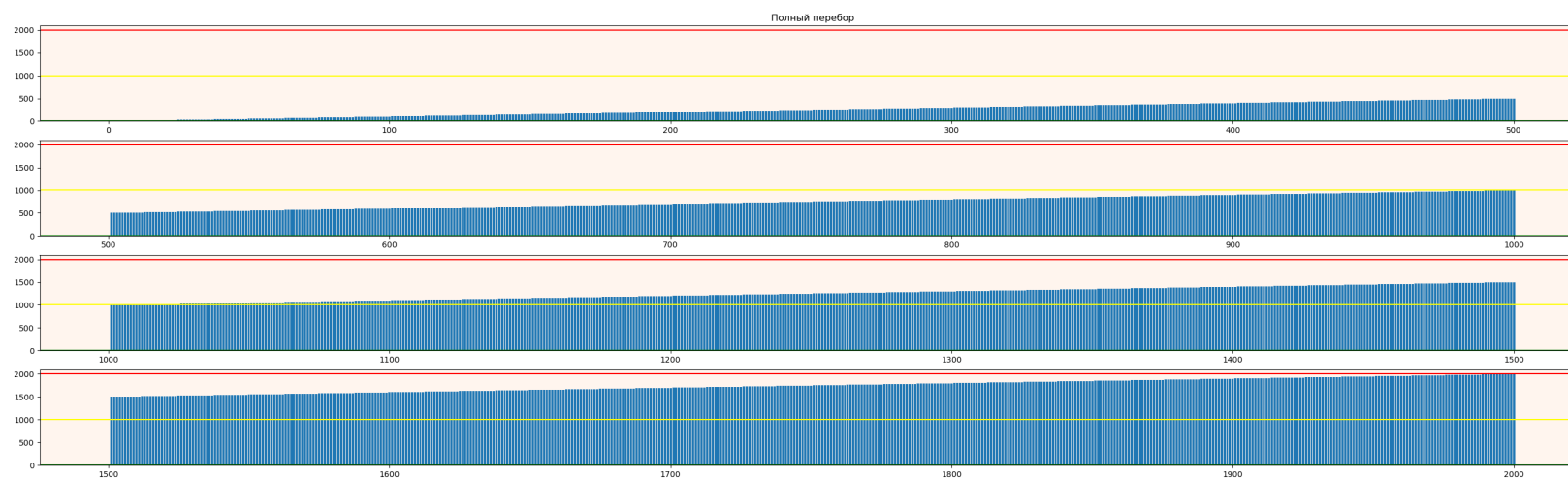


Рисунок 4.2 — Полный перебор, гистограмма типа 1

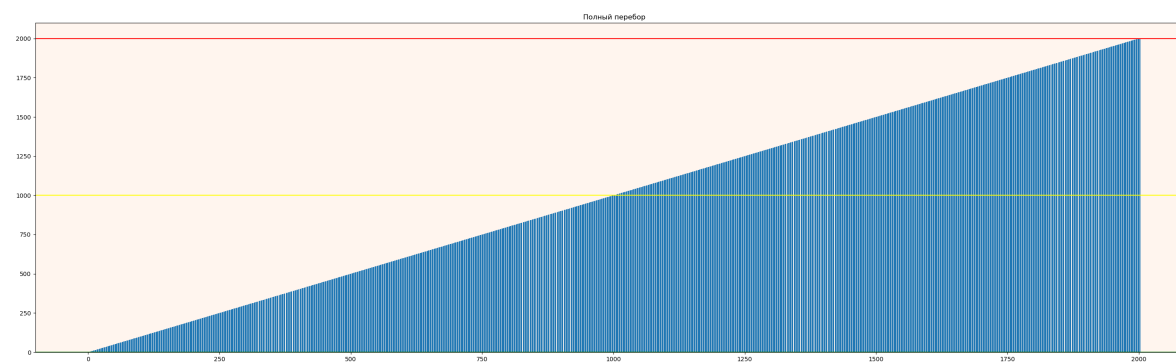


Рисунок 4.3 — Полный перебор, гистограмма типа 2

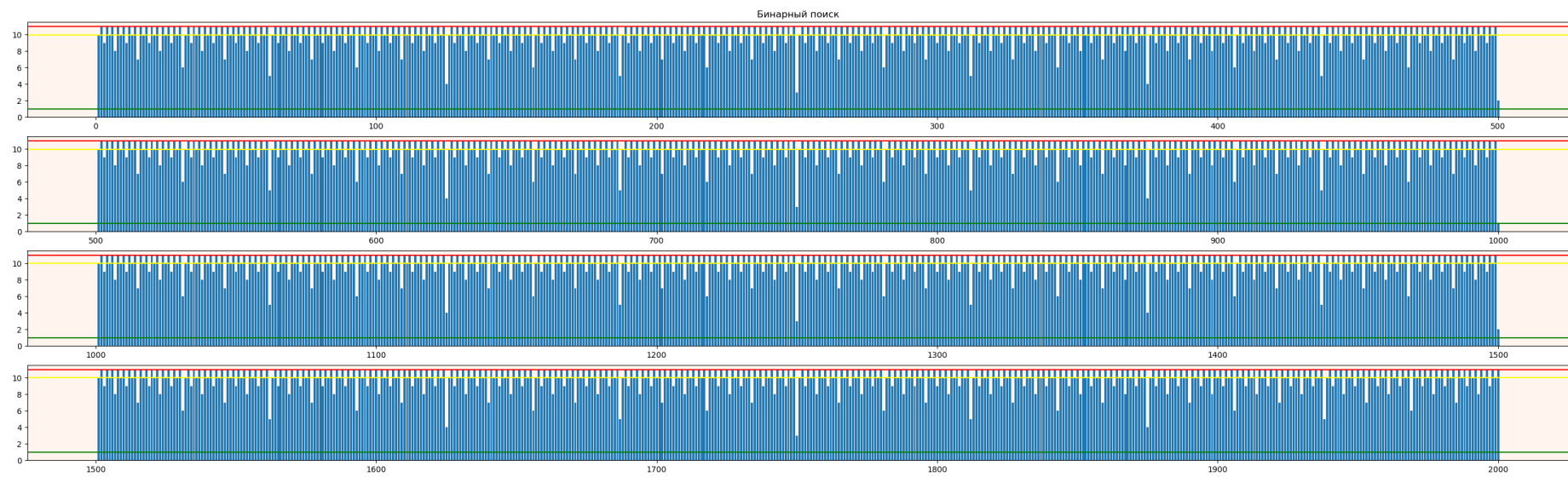


Рисунок 4.4 — Бинарный поиск, гистограмма типа 1

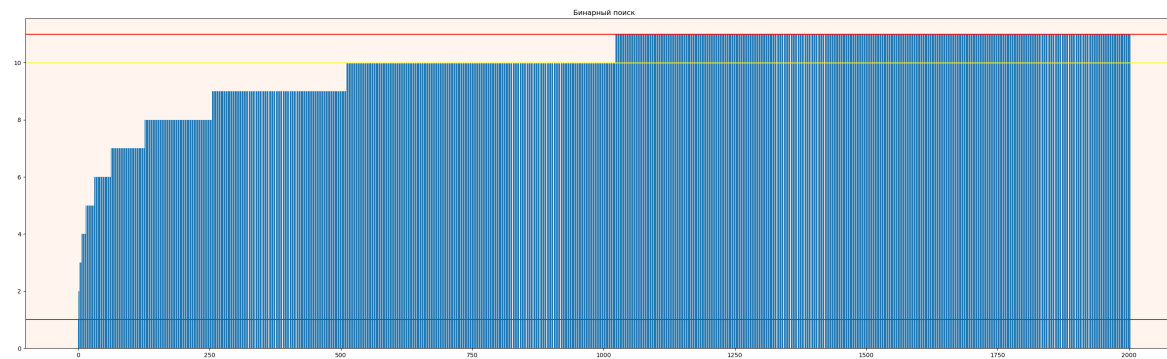


Рисунок 4.5 — Бинарный поиск, гистограмма типа 2

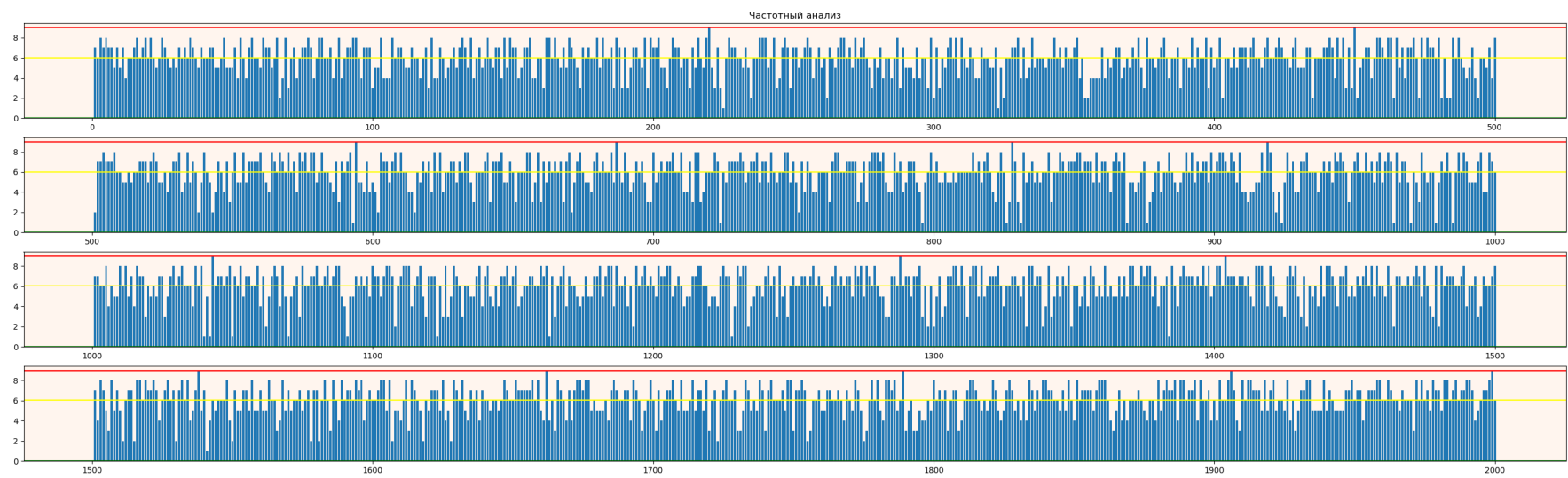


Рисунок 4.6 — Частотный анализ, гистограмма типа 1

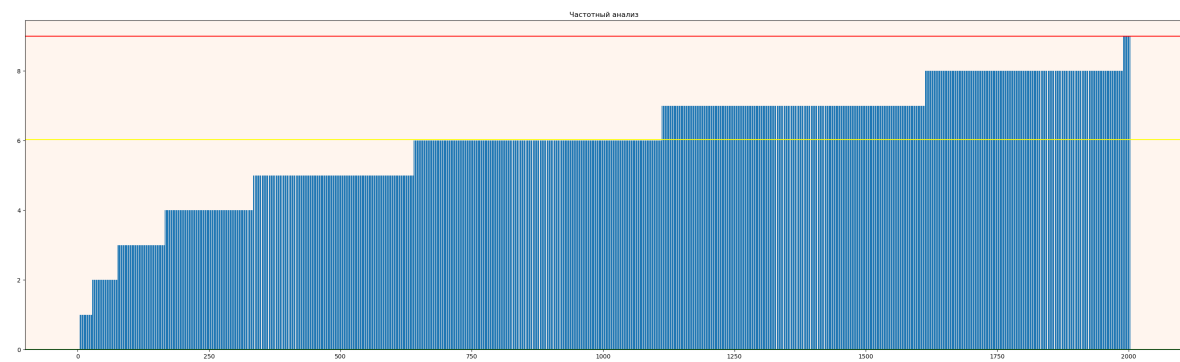


Рисунок 4.7 — Частотный анализ, гистограмма типа 2

4.5 Вывод

Как видно из графиков, частотный анализ требует в среднем требует в 1.5-2 раза меньше сравнений. Как и ожидалось, алгоритм полного перебора требует наибольшего количества сравнений из всех представленных алгоритмов.

Также видно, что гистограмма типа 1 для полного перебора - возрастающая последовательность, в то время как для алгоритма бинарного поиска и, в частности, алгоритма частотного анализа, последовательность не является монотонно возрастающей в виду идеи работы алгоритма бинарного поиска.

Заключение

Как видно из графиков, частотный анализ требует в среднем требует в 1.5-2 раза меньше сравнений. Как и ожидалось, алгоритм полного перебора требует наибольшего количества сравнений из всех представленных алгоритмов.

Также видно, что гистограмма типа 1 для полного перебора - возрастающая последовательность, в то время как для алгоритма бинарного поиска и, в частности, алгоритма частотного анализа, последовательность не является монотонно возрастающей в виду идеи работы алгоритма бинарного поиска.

На основании проделанной работы можно сделать следующий вывод: чем больше размер словаря, тем более рационально использовать эффективные алгоритмы поиска. Однако стоит помнить, что бинарный поиск применим только к отсортированным данным, поэтому может возникнуть ситуация, когда поддержание упорядоченной структуры занимает нерационально большое время и преимущества от бинарного поиска нивелируются.

Список использованных источников

1. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ. — М.В. Ульянов, 2007.
2. Новый алгоритм частотного анализа. — Тутыгин В.С., Дебелова А.В., 2009.
3. Ubuntu. — [Электронный ресурс]. Режим доступа: <https://ru.wikipedia.org/wiki/Ubuntu>., 16.09.2021.
4. Linux. — [Электронный ресурс]. Режим доступа: <https://ru.wikipedia.org/wiki/Linux>., 16.09.2021.
5. Процессор AMD Ryzen(TM) 5. — [Электронный ресурс]. Режим доступа: <https://shop.lenovo.ru/product/81YM007FRU/>., 21.09.2021.