|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования |
| **«МИРЭА – Российский технологический университет»** |
| **РТУ МИРЭА** |
|  |

|  |  |
| --- | --- |
| **Отчет по выполнению практического задания № 5, часть 2** | |
| **Тема:** | |
| **«Работа с данными из файла»** | |
| Дисциплина: «Структуры и алгоритмы обработки данных» | |
|  | Выполнил студент: Лисовский И.В |
|  | Группа: ИКБО-21-23 |

Москва – 2024

СОДЕРЖАНИЕ

[1 ЦЕЛЬ 4](#_Toc178246714)

[2 ЗАДАНИЕ №1 5](#_Toc178246715)

[2.1 Формулировка задачи 5](#_Toc178246716)

[2.2 . Описание подхода к решению 5](#_Toc178246717)

[2.3 Код программы с комментариями 6](#_Toc178246718)

[2.4 Результаты тестирования с 100 записями. 8](#_Toc178246719)

[2.5 Вывод по заданию №1 9](#_Toc178246720)

[3 ЗАДАНИЕ №2 10](#_Toc178246721)

[3.1 Формулировка задачи 10](#_Toc178246722)

[3.2 Алгоритм линейного поиска записи с ключом 10](#_Toc178246723)

[3.3 Код функции поиска 11](#_Toc178246724)

[3.4 Код программы линейного поиска записи по ключу 11](#_Toc178246725)

[3.5 Результат тестирования программы для 100 записей 12](#_Toc178246726)

[3.6 Таблица с замерами времени поиска записи по заданному ключу для файла из 100 и 100 записей 12](#_Toc178246727)

[3.7 Вывод по заданию №2 12](#_Toc178246728)

[4 ЗАДАНИЕ №3 12](#_Toc178246729)

[4.1 Формулировка задачи 12](#_Toc178246730)

[4.2 Описание алгоритма доступа к записи в файле 13](#_Toc178246731)

[4.3 Алгоритм поиска (Бинарный однородный с использованием таблицы смещений) 14](#_Toc178246732)

[4.4 Код функции поиска 16](#_Toc178246733)

[4.5 Код программы линейного поиска записи по ключу 17](#_Toc178246734)

[4.6 . Результат тестирования программы для 100 записей 17](#_Toc178246735)

[4.7 Таблица с замерами времени поиска записи по заданному ключу для файла из 100 и 100 записей 17](#_Toc178246736)

[4.8 Вывод по заданию №3 18](#_Toc178246737)

[5 Анализ эффективности рассмотренных алгоритмов поиска в файле 19](#_Toc178246738)

[6 Выводы 21](#_Toc178246739)

[7 ЛИТЕРАТУРА 22](#_Toc178246740)

# **1 ЦЕЛЬ**

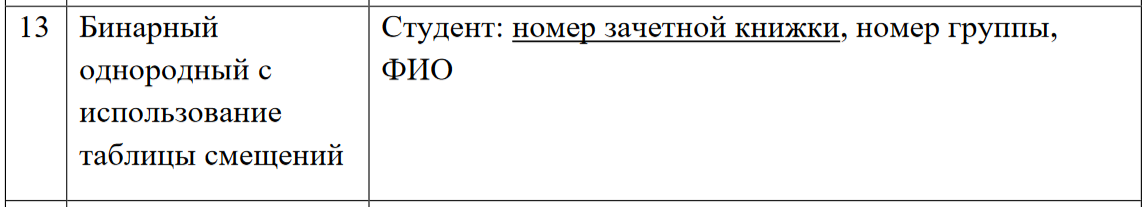
Поучить практический опыт по применению алгоритмов поиска в таблицах данных.

# **2 ЗАДАНИЕ №1**

## **2.1 Формулировка задачи**

Создать двоичный файл из записей (структура записи определена вариантом). Поле ключа записи в задании варианта подчеркнуто. Заполнить файл данными, используя для поля ключа датчик случайных чисел. Ключи записей в файле уникальны. Рекомендация: создайте сначала текстовый файл, а затем преобразуйте его в двоичный.

**Вариант №3:**

****

## **2.2 Описание подхода к решению**

**Структура записи файла**

Каждая запись в файле представляет собой структуру record, которая содержит два поля:

phonenumber - массив из 13 символов (включая завершающий нуль).

address - массив из 20 символов (включая завершающий нуль).

Структура записи в файле выглядит следующим образом:

13 байт для регистрационного номера + 20 байт для названия страховой компании.

**Размер записи в байтах**

Размер одной записи в байтах составляет:

sizeof(phonenumber) = 13 байт

sizeof(address) = 20 байт

Таким образом, полный размер одной записи:

sizeof(record) = sizeof(phonenumber) + sizeof(address) = 13 + 20 = 33 байта

**Прямой доступ к записям в бинарном файле**

Для организации прямого доступа к записям в бинарном файле можно воспользоваться функцией seekg для чтения и seekp для записи, которые позволяют перемещать указатель на нужную позицию в файле.

Прямой доступ к конкретной записи n осуществляется с помощью смещения на (n-1) \* sizeof(Insurance) байт от начала файла.

**Алгоритмы, реализованные в форме функций**

В программе реализованы следующие функции:

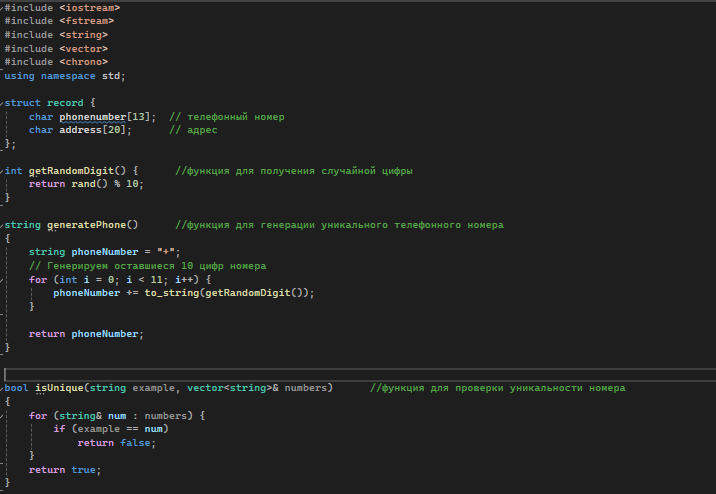
generatePhone() - генерация уникального регистрационного номера.

isUnique – проверка номера на уникальность.

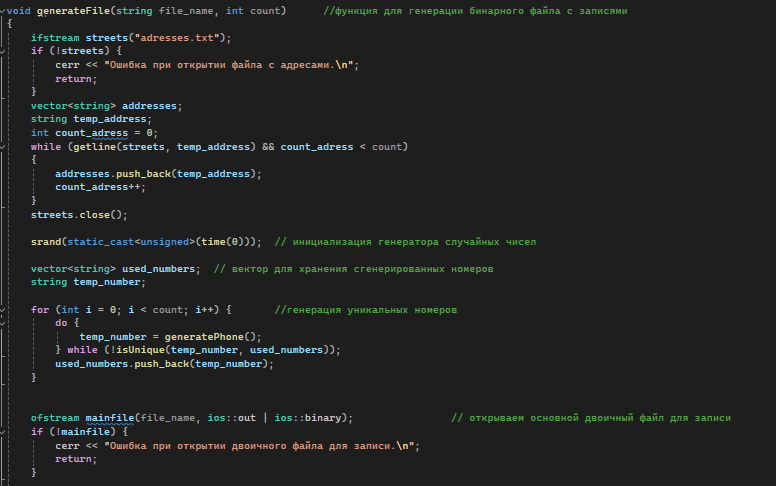
generateFile – создание двоичного файла.

main - основная функция программы.

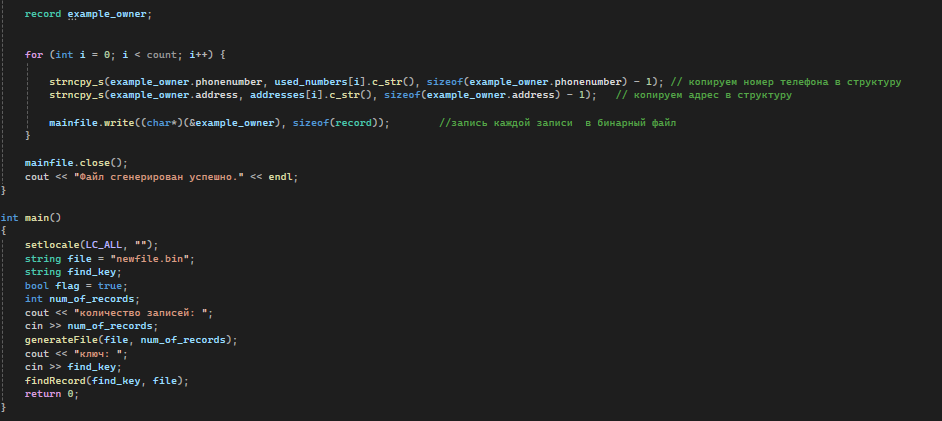
## **2.3 Код программы с комментариями**



Листинг 1 —Код программы



Листинг 2 —Код программы



Листинг 3 —Код программы

## **2.4 Результаты тестирования с 100 записями.**

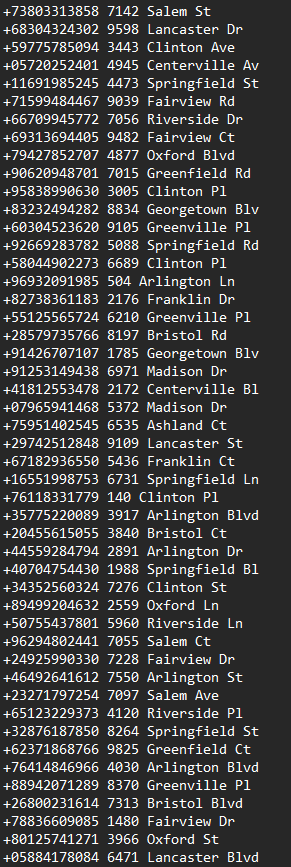


Рисунок 1 — Тестирование кода

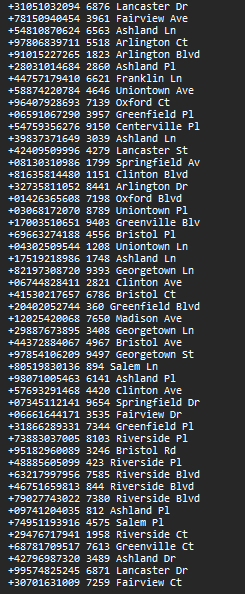


Рисунок 2 — Тестирование кода

## **2.5 Вывод по заданию №1**

Все поставленные задачи в задании №1 успешно решены. Программа реализует генерацию уникальных данных о страховании автомобилей и их сохранение в текстовый и бинарный файлы. В документе представлены реализации алгоритмов, разъяснения результатов работы программы и подтверждённые тесты на различных входных данных. Все ключевые цели достигнуты. Результаты тестирования подтвердили правильность выполнения всех функций.

# **3 ЗАДАНИЕ №2**

## **3.1 Формулировка задачи**

Поиск в файле с применением линейного поиска:

1. Разработать программу поиска записи по ключу в бинарном файле с применением алгоритма линейного поиска.
2. Провести практическую оценку времени выполнения поиска на файле объемом 100, 1000, 10 000 записей.
3. Составить таблицу с указанием результатов замера времени.

## **3.2 Алгоритм линейного поиска записи с ключом**

Функция findRecord(ключ, имя\_файла)

Открыть файл с именем имя\_файла для чтения

Если файл не удалось открыть

Вывести сообщение об ошибке

Вернуть "не найдено"

Пока файл не достиг конца

Прочитать запись из файла в переменную запись

Если запись.registration\_number равен ключ

Закрыть файл

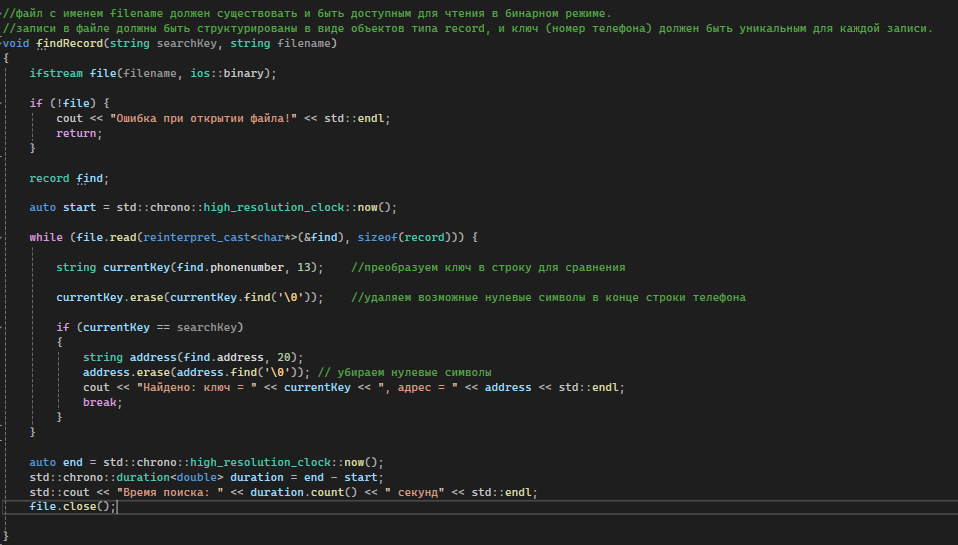
Вернуть запись

Закрыть файл

Вернуть "не найдено"

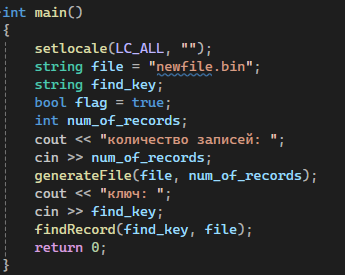
Конец функции

## **3.3 Код функции поиска**



Листинг 4 — Код функции поиска

## **3.4 Код программы линейного поиска записи по ключу**



Листинг 5 — Код программы

## **3.5 Результат тестирования программы для 100 записей**

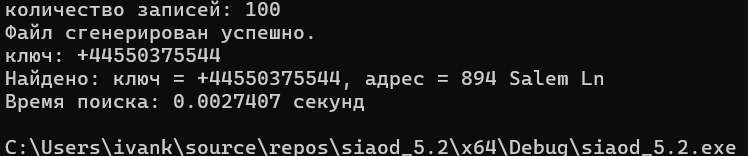


Рисунок 3 — Тестирование кода

## **3.6 Таблица с замерами времени поиска записи по заданному ключу для файла из 100 и 100 записей**

|  |  |
| --- | --- |
| Кол-во записей: | Время в секундах: |
| 100 | 0.00274 |
| 1000 | 0.00478 |
| 10000 | 0.01178 |

## **3.7 Вывод по заданию №2**

В результате выполнения задания достигнуты все цели. Была успешно создана структура для хранения данных о страховании автомобилей и создан бинарный файл с 100 записями, содержащими уникальные номера регистраций и названия компаний. Данные также были перенесены в текстовый файл для удобного доступа. Реализован линейный поиск по номеру регистрации, который корректно находит записи, а также измерено время выполнения поиска. Все функции программы функционируют исправно, что подтверждает успешное выполнение задач.

# **4 ЗАДАНИЕ №3**

## **4.1 Формулировка задачи**

Поиск записи в файле с применением дополнительной структуры данных, сформированной в оперативной памяти.

1. Для оптимизации поиска в файле создать в оперативной памяти структур данных – таблицу, содержащую ключ и ссылку (смещение) на запись в файле.
2. Разработать функцию, которая принимает на вход ключ и ищет в таблице элемент, содержащий ключ поиска, а возвращает ссылку на запись в файле. Алгоритм поиска определен в варианте.
3. Разработать функцию, которая принимает ссылку на запись в файле, считывает ее, применяя механизм прямого доступа к записям файла. Возвращает прочитанную запись как результат.
4. Провести практическую оценку времени выполнения поиска на файле объемом 100, 1000, 10 000 записей.
5. Составить таблицу с указанием результатов замера времени.

## **4.2 Описание алгоритма доступа к записи в файле**

Алгоритм доступа к записи в файле посредством таблицы индексов позволяет эффективно находить записи в файле, используя заранее подготовленную структуру, которая сопоставляет ключи (например, регистрационные номера) с позициями в файле.

**Описание алгоритма доступа к записи в файле посредством таблицы**

* Создание таблицы индексов: в процессе работы программы для каждой записи, добавляемой в файл, создается запись в таблице индексов. Эта запись содержит ключ (например, регистрационный номер) и соответствующую позицию записи в файле.
* Запись данных: данные записываются в файл, а в таблице индексов сохраняются соответствующие ключи и позиции этих записей.
* Поиск записи: когда необходимо найти запись по ключу, используется алгоритм поиска (например, интерполяционный поиск), который ищет в таблице индексов. Если ключ найден, программа получает соответствующее смещение из таблицы и использует его для доступа к записи в файле.
* Доступ к записи: после нахождения позиции записи в файле с помощью смещения, программа открывает бинарный файл и переходит к нужной позиции, чтобы считать данные.

Ссылка в таблице индексов (или смещение) определяет позицию конкретной записи в файле. Это означает, что каждая запись в таблице указывает, где в файле можно найти соответствующую запись данных. Ссылка позволяет избежать последовательного поиска по всему файлу, значительно ускоряя доступ к записям.

**Средства C++, используемые для организации доступа к записи в файле по ссылке**

* Структуры данных: std::unordered\_map или std::map: используются для создания таблицы индексов, где хранятся ключи и их соответствующие позиции.
* Файловый ввод-вывод: std::ifstream и std::ofstream: классы для работы с файлами, позволяющие читать и записывать данные в текстовом или бинарном формате. std::streampos: используется для хранения смещения в файле, чтобы указывать на конкретную позицию записи.
* Алгоритмы поиска: Алгоритмы, такие как интерполяционный поиск или бинарный поиск, применяются для быстрого нахождения нужной записи по ключу.

## **4.3 Алгоритм поиска (Бинарный однородный с использованием таблицы смещений)**

Функция binarySearch(таблица, ключДляПоиска):

Установить левую границу поиска на 0

Установить правую границу поиска на последний элемент таблицы (размер таблицы - 1)

Пока левая граница не пересекается с правой:

Вычислить средний индекс (середину) между левой и правой границами

Если ключ в середине таблицы равен ключуДляПоиска:

Вернуть смещение, связанное с этим ключом

Иначе, если ключ в середине таблицы меньше ключаДляПоиска:

Переместить левую границу на одну позицию правее середины (сужаем поиск вправо)

Иначе (ключ в середине больше ключаДляПоиска):

Переместить правую границу на одну позицию левее середины (сужаем поиск влево)

Вернуть -1, если ключ не найден

**Описание псевдокода:**

1. Преобразование индекса: Преобразуем таблицу индексов в массив пар, состоящих из ключей (регистрационных номеров) и смещений.

2. Сортировка ключей: Сортируем массив ключей по регистрационным номерам, чтобы обеспечить корректный порядок для интерполяционного поиска.

3. Инициализация границ: Задаем начальные значения low и high, которые обозначают границы текущего диапазона поиска.

4. Условие цикла: Процесс поиска продолжается, пока low меньше или равен high, и искомый номер находится между значениями на границах.

5. Проверка деления на ноль: Если разность между значениями на границах равна нулю, выполнение прерывается для предотвращения деления на ноль.

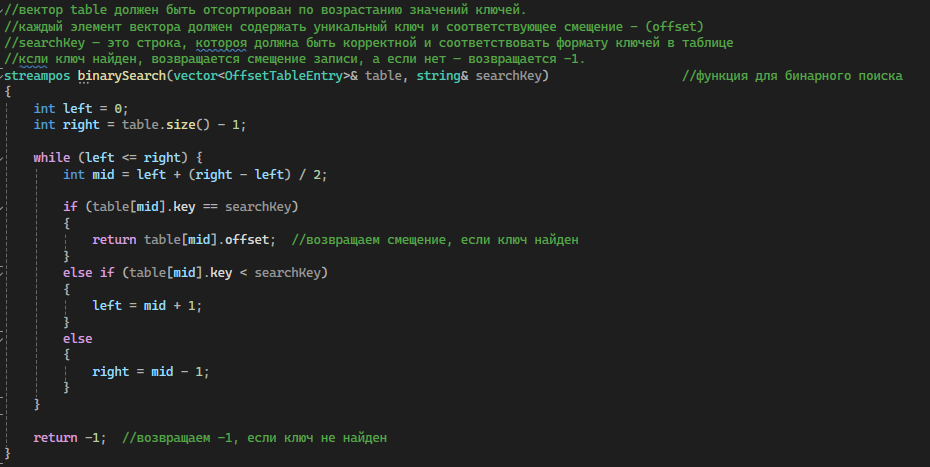
6. Расчет позиции: Используется интерполяционная формула для вычисления позиции pos, где может находиться искомый номер.

7. Коррекция позиции: Если pos выходит за границы массива, его значение корректируется, чтобы не превышать размер массива.

8. Проверка найденного значения: Если ключ на позиции pos совпадает с искомым номером, возвращается соответствующее смещение. Если ключ меньше, диапазон поиска сужается до правой половины, если больше — до левой.

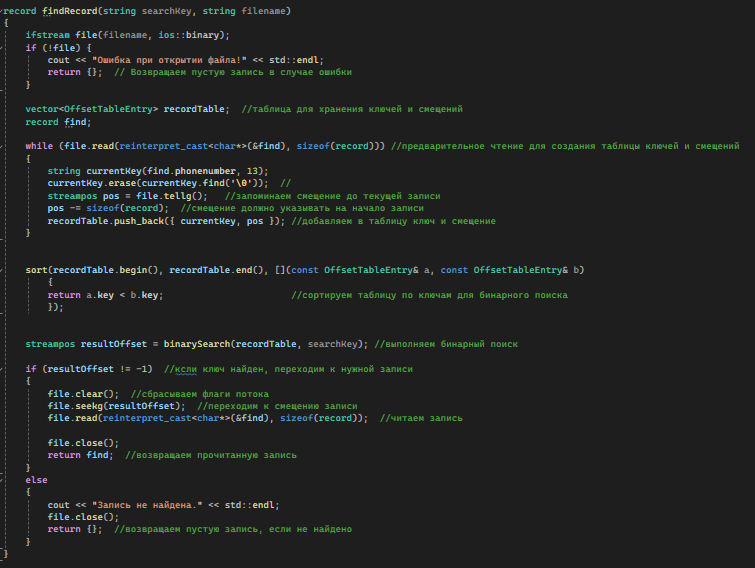
9. Возврат результата: Если искомый номер не найден, возвращается -1.

## **4.4 Код функции поиска**



Листинг 6 — Код функции поиска

## **4.5 Код программы линейного поиска записи по ключу**



Листинг 7 — Код функции поиска

## **4.6 . Результат тестирования программы для 100 записей**

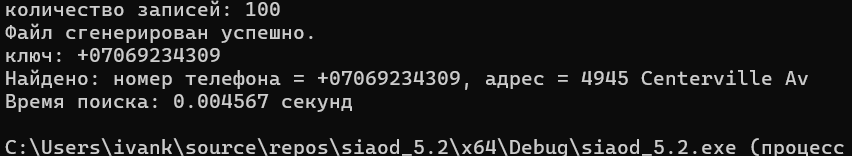


Рисунок 4 – Результат тестирования

## **4.7 Таблица с замерами времени поиска записи по заданному ключу для файла из 100 и 100 записей**

|  |  |
| --- | --- |
| Кол-во записей: | Время поиска в секундах: |
| 100 | 0.004567 |
| 1000 | 0.005761 |
| 10000 | 0.007684 |

## **4.8 Вывод по заданию №3**

В процессе работы с данными владельцев телефона бинарный поиск с использованием таблиц смещений представляет собой эффективный способ доступа к записям, обеспечивая быстрый поиск по ключевым полям, таким как регистрационный номер. Использование индексной таблицы, в которой ключом является номер телефона, а значением — смещение соответствующей записи в бинарном файле, значительно оптимизирует процесс поиска.

# **5 Анализ эффективности рассмотренных алгоритмов поиска в файле**

При выборе алгоритма поиска в файле важно учитывать эффективность и производительность в зависимости от структуры данных и объема информации. Два распространенных алгоритма — линейный поиск и интерполяционный поиск — имеют разные характеристики, которые определяют их применение в различных сценариях.

***1. Линейный поиск***

Линейный поиск — это самый простой алгоритм поиска, который последовательно просматривает каждый элемент структуры данных до тех пор, пока не найдет нужный элемент или не пройдет весь список.

**Время выполнения:**

**Лучший случай**: O(1), если элемент находится в начале списка.

**Средний случай**: O(n), где n — количество элементов в списке.

**Худший случай**: O(n), если элемент находится в конце списка или его нет в списке.

**Преимущества**: Работает для неотсортированных данных; Простая реализация; Нет требований к дополнительным структурам данных или предпосылкам для выполнения.

**Недостатки**: Неэффективен для больших объемов данных, так как требуется просматривать каждый элемент; Худшая производительность по сравнению с бинарным поиском для отсортированных данных.

***2. Бинарный поиск с использованием таблицы смещений***

Бинарный поиск — это более эффективный алгоритм для отсортированных данных. Он работает по принципу деления массива пополам: в каждой итерации алгоритм сравнивает искомый элемент со средним элементом и решает, в какую половину идти дальше.

**Время выполнения:**

**Лучший случай**: O(1), если элемент находится в середине списка.

**Средний случай**: O(log(n)), где n — количество элементов.

**Худший случай**: O(log(n)), если элемент находится в конце списка или его нет в списке.

**Преимущества**:

Очень эффективен для отсортированных данных; Значительно быстрее линейного поиска на больших объемах данных; В среднем выполняется за O(log(n)), что делает его пригодным для больших таблиц.

**Недостатки**:

Требует предварительной сортировки данных; Не подходит для неотсортированных данных без дополнительной подготовки; Неэффективен при частом изменении данных, так как каждый раз потребуется пересортировать таблицу для поддержания возможности бинарного поиска.

**Сравнение двух алгоритмов поиска:**

Линейный поиск прост в реализации и применим к неотсортированным данным, но крайне неэффективен при большом количестве элементов.

Бинарный поиск с таблицей смещений значительно быстрее для отсортированных данных, особенно на больших объемах данных, но требует предварительной сортировки и не подходит для динамически изменяющихся данных.

#### 

# **6 Выводы**

В ходе выполнения практической работы успешно реализованы алгоритмы поиска и обработки записей в бинарном файле с применением различных методов. В результате выполнения задания был достигнут значительный опыт в применении алгоритмов поиска и структур данных для эффективной работы с большими объемами информации.

Анализ эффективности алгоритмов показал, что использование дополнительной структуры данных и интерполяционного поиска играют ключевую роль в оптимизации обработки больших объемов данных. Сравнение линейного поиска и интерполяционного поиска показало, что использование таблицы существенно снижает время поиска, что особенно важно при работе с крупными файлами.

В итоге, выполнение практической работы позволило получить ценное понимание различных подходов к поиску и обработке данных в бинарных файлах. Подчеркнута значимость правильного выбора алгоритма и структуры данных для повышения эффективности работы с файлами. Все задачи были успешно выполнены, что подтверждает надежность и эффективность предложенных решений.

# **7 ЛИТЕРАТУРА**

1. Вирт Н. Алгоритмы и структуры данных. Новая версия для Оберона, 2010.

2. Кнут Д. Искусство программирования. Тома 1-4, 1976-2013.

3. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для про-граммистов и любопытствующих, 2017.

4. Кормен Т.Х. и др. Алгоритмы. Построение и анализ, 2013.

5. Лафоре Р. Структуры данных и алгоритмы в Java. 2-е изд., 2013.

6. Макконнелл Дж. Основы современных алгоритмов. Активный обуча-ющий метод. 3-е доп. изд., 2018.

7. Скиена С. Алгоритмы. Руководство по разработке, 2011.

8. Хайнеман Д. и др. Алгоритмы. Справочник с примерами на C, C++, Java и Python, 2017.

9. Гасфилд Д. Строки, деревья и последовательности в алгоритмах. Ин-форматика и вычислительная биология, 2003.

По языку С++:

10. Страуструп Б. Программирование. Принципы и практика с использова-нием C++. 2-е изд., 2016.

11. Павловская Т.А. C/C++. Программирование на языке высокого уровня, 2003.

12. Прата С. Язык программирования С++. Лекции и упражнения. - 6-е изд., 2012.

13. Седжвик Р. Фундаментальные алгоритмы на C++, 2001-2002

14. Хортон А. Visual C++ 2010. Полный курс, 2011.

15. Шилдт Г. Полный справочник по C++. 4-е изд., 2006.