МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 5.2

по дисциплине
«Структуры и алгоритмы обработки данных»

Тема: «Алгоритмы поиска в таблице при работе с данными из файла»

Выполнил студент группы группа ИКБО-10-23 Лазаренко С.А.

Приняла доцент кафедры Макеева О.В.

Москва 2024

ОГЛАВЛЕНИЕ

ЦЕЛЬ РАБОТЫ

Поучить практический опыт по применению алгоритмов поиска в таблицах данных. Сделать выводы о проделанной работе, основанные на полученных результатах.

ЗАДАНИЕ №1

1.1 Постановка задания 1

Целью задания является создание двоичного файла, содержащего записи со структурой, определенной в соответствии с вариантом. Поле ключа записи (номер читательского билета) должно быть заполнено случайными уникальными числами. Рекомендуется сначала создать текстовый файл, а затем преобразовать его в двоичный.

1.2 Описание подхода к решению задания 1

Структура записи определяется следующим образом:

- Номер читательского билета (ключ) — целое пятизначное число int.
- ФИО — строка, максимальная длина 50 символов.
- Адрес — строка, максимальная длина 100 символов.

```
struct Record {
    int cardNumber;
    char name[50];
    char address[100];
};
```

Рисунок 1 – Структура записи

Размер записи в байтах будет равен сумме:

- 4 байта на хранение целого числа (int).
- 50 байт на ФИО.
- 100 байт на адрес.

Прямой доступ организуется с использованием смещения, вычисляемого по формуле: Смещение = номер записи * размер записи. Используя seekg и seekp, можно позиционировать указатель файла для чтения или записи на нужную позицию.

Функция generateData заполняет структуру данных случайными уникальными номерами и случайными строками для полей "ФИО" и "Адрес". Сначала генерируются уникальные номера читательских билетов (ключи), затем к каждому ключу добавляются случайные строки, представляющие имя и адрес.

```
void generateData(int recordCount, vector<Record>& records) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> distr(10000, 99999); // Генерация пятизначных чисел
    set<int> uniqueNumbers;

    // Массивы возможных ФИО и адресов
    vector<string> names = { "Ivan Ivanov", "Petr Petrov", "Sergey Sidorov", "Anna Smirnova", "Olga Pavlova"
        };
    vector<string> addresses = { "Moscow, Lenin St.", "Saint Petersburg, Nevskiy Ave.", "Kazan, Tverskaya
        St.", "Novosibirsk, Karl Marx St.", "Yekaterinburg, Gagarina Ave." };

    while (uniqueNumbers.size() < recordCount) {
        uniqueNumbers.insert(distr(gen)); // Уникальные номера
    }

    auto it = uniqueNumbers.begin();
    for (int i = 0; i < recordCount; ++i) {
        Record rec;
        rec.cardNumber = *it++;
        strncpy(rec.name, names[i % names.size()].c_str(), sizeof(rec.name) - 1);
        strncpy(rec.address, addresses[i % addresses.size()].c_str(), sizeof(rec.address) - 1);
        rec.name[sizeof(rec.name) - 1] = '\0';
        rec.address[sizeof(rec.address) - 1] = '\0';
        records.push_back(rec);
    }
}
```

Рисунок 2 – код функции generateData

Функция writeTextFile записывает вектор структур записей в текстовый файл, где каждая запись представлена строкой, содержащей значения полей, разделенных запятыми.

Открывается файл с заданным именем для записи. Для каждой записи из вектора records создается строка в формате "номер читательского билета, ФИО, Адрес". Строки записываются в текстовый файл построчно.

```
void writeTextFile(const string& filename, const vector<Record>& records) {
    ofstream outFile(filename);
    if (!outFile) {
        cerr << "Ошибка при открытии файла " << filename << endl;
        return;
    }

    for (const auto& rec : records) {
        outFile << rec.cardNumber << "\t" << rec.name << "\t" << rec.address << endl;
    }
    outFile.close();
}
```

Рисунок 3 – код функции writeTextFile

Функция converToBinary преобразует текстовый файл, содержащий записи, в двоичный формат.

Открывается текстовый файл для чтения. Для каждой строки из текстового файла извлекаются значения полей и преобразуются в структуру Record.

Структуры записываются в выходной двоичный файл с использованием функции write.

```
void convertToBinary(const string& textFilename, const string& binaryFilename) {
    ifstream inFile(textFilename);
    ofstream outFile(binaryFilename, ios::binary);

    if (!inFile || !outFile) {
        cerr << "Ошибка при открытии файлов" << endl;
        return;
    }

    Record rec;
    while (inFile >> rec.cardNumber) {
        inFile.ignore(); // Пропускаем табуляцию
        inFile.getline(rec.name, sizeof(rec.name), '\t');
        inFile.getline(rec.address, sizeof(rec.address));
        outFile.write(reinterpret_cast<char*>(&rec), sizeof(Record));
    }

    inFile.close();
    outFile.close();
}
```

Рисунок 4 – код функции convertToBinary

Функция readBinaryFile считывает и выводит на экран содержимое двоичного файла.

Открывается двоичный файл для чтения. Последовательно считываются записи до конца файла и выводятся на экран. Каждая запись считывается как структура Record и отображается в читаемом виде.

```
void readBinaryFile(const string& binaryFilename) {
    ifstream inFile(binaryFilename, ios::binary);
    if (!inFile) {
        cerr << "Ошибка при открытии файла " << binaryFilename << endl;
        return;
    }

    Record rec;
    while (inFile.read(reinterpret_cast<char*>(&rec), sizeof(Record))) {
        cout << "Номер читательского билета: " << rec.cardNumber << "\n"
             << "ФИО: " << rec.name << "\n"
             << "Адрес: " << rec.address << "\n"
             << "-------------------------" << endl;
    }

    inFile.close();
}
```

Рисунок 5 – код функции readBinaryFile

Предусловия и постусловия для функций:

1. Функция generateData

Предусловие: *recordCount* > 0, вектор *records* пустой или готов к записи.

Постусловие: вектор *records* содержит *recordCount* записей со случайными данными.

2. Функция writeTextFile

Предусловие: файл *filename* должен быть доступен для записи.

Постусловие: данные из вектора *records* записаны в текстовый файл.

3. Функция convertToBinary

Предусловие: текстовый файл *textFilename* существует и доступен для чтения.

Постусловие: создан двоичный файл *binaryFilename* с данными.

4. Функция readBinaryFile

Предусловие: двоичный файл *binaryFilename* существует и доступен для чтения.

Постусловие: данные из файла выведены на экран.

1.3 Код программы на C++

```
int main() {
    const int recordCount = 100;
    vector<Record> records;

    // Генерация данных
    generateData(recordCount, records);

    // Запись данных в текстовый файл
    const string textFilename = "records.txt";
    writeTextFile(textFilename, records);

    // Конвертация текстового файла в двоичный
    const string binaryFilename = "records.bin";
    convertToBinary(textFilename, binaryFilename);

    // Чтение данных из двоичного файла
    cout << "Чтение данных из двоичного файла:" << endl;
    readBinaryFile(binaryFilename);

    return 0;
}
```

Рисунок 6 – код функции main

1.4 Тестирование программы

Протестируем работу программы для 100 записей и убедимся в корректности кода:

```
Номер читательского билета: 70329
ФИО: Anna Smirnova
Адрес: Novosibirsk, Karl Marx St.
-------------------------
Номер читательского билета: 71653
ФИО: Olga Pavlova
Адрес: Yekaterinburg, Gagarina Ave.
-------------------------
Номер читательского билета: 72173
ФИО: Ivan Ivanov
Адрес: Moscow, Lenin St.
-------------------------
Номер читательского билета: 72193
ФИО: Petr Petrov
Адрес: Saint Petersburg, Nevskiy Ave.
-------------------------
Номер читательского билета: 72475
ФИО: Sergey Sidorov
Адрес: Kazan, Tverskaya St.
-------------------------
Номер читательского билета: 73370
ФИО: Anna Smirnova
Адрес: Novosibirsk, Karl Marx St.
-------------------------
Номер читательского билета: 73881
ФИО: Olga Pavlova
Адрес: Yekaterinburg, Gagarina Ave.
-------------------------
Номер читательского билета: 73906
ФИО: Ivan Ivanov
Адрес: Moscow, Lenin St.
-------------------------
Номер читательского билета: 75424
ФИО: Petr Petrov
Адрес: Saint Petersburg, Nevskiy Ave.
-------------------------
Номер читательского билета: 76581
ФИО: Sergey Sidorov
Адрес: Kazan, Tverskaya St.
-------------------------
```

Рисунок 7 – Часть результата работы программы

ЗАДАНИЕ №2

2.1 Постановка задания 2

Разработать программу для поиска записи по ключу в бинарном файле с использованием алгоритма линейного поиска. Необходимо провести практическую оценку времени выполнения поиска на файлах объемом 100, 1000 и 10,000 записей, а также представить результаты в таблице.

2.2 Алгоритм программы

Рассмотрим псевдокод линейного поиска:

```
Function LinearSearch(FileName, key):
    Open FileName for reading
    If file is not open:
        Print "Error opening file"
        Exit

    While not end of file:
        Read record from file

        If record.cardNumber equals key:
            Close file
            Return record

    Close file
    Return "Record not found"
```

Проверяет, был ли файл успешно открыт, и выводит ошибку в случае неудачи. Проходит по записям в файле, пока не достигнут конец файла. Сравнивает номер читательского билета в текущей записи с искомым ключом. Закрывает файл после завершения работы. Возвращает найденную запись или сообщение о том, что запись не найдена.

2.3 Код функции поиска

Функция линейного поиска последовательно открывает бинарный файл и читает каждую запись структуры *Record* одну за другой, сравнивая значение *cardNumber* каждой записи с искомым ключом *searchKey*. Если запись с совпадающим *cardNumber* найдена, она возвращается как результат поиска, и функция завершает работу.

Рассмотрим код функции поиска:

```
Record linearSearch(const string& binaryFilename, int key) {
    ifstream inFile(binaryFilename, ios::binary);
    if (!inFile) {
        cerr << "Ошибка при открытии файла " << binaryFilename << endl;
        throw runtime_error("File not found");
    }

    Record rec;
    while (inFile.read(reinterpret_cast<char*>(&rec), sizeof(Record))) {
        if (rec.cardNumber == key) {
            inFile.close();
            return rec;
        }
    }

    inFile.close();
    throw runtime_error("Запись не найдена");
}
```

Рисунок 8 – Код функции поиска

2.4 Код программы линейного поиска записи по ключу

Используется генератор случайных чисел *mt19937* для инициализации генерации случайных номеров читательских билетов (в диапазоне от 10000 до 99999*). uniqueNumbers (set<int>*) хранит уникальные номера, чтобы избежать дублирования. Номера генерируются и добавляются в множество *uniqueNumbers* до тех пор, пока не будет достигнуто требуемое количество *recordCount.*

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <random>
#include <set>
#include <cstring>
#include <chrono>
#include <stdexcept>

using namespace std;

struct Record {
    int cardNumber;
    char name[50];
    char address[100];
};

void generateData(int recordCount, vector<Record>& records) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> distr(10000, 99999);
    set<int> uniqueNumbers;

    vector<string> names = { "Ivan Ivanov", "Petr Petrov", "Sergey Sidorov", "Anna Smirnova", "Olga Pavlova"
        };
    vector<string> addresses = { "Moscow, Lenin St.", "Saint Petersburg, Nevskiy Ave.", "Kazan, Tverskaya
        St.", "Novosibirsk, Karl Marx St.", "Yekaterinburg, Gagarina Ave." };

    // Генерация уникальных номеров
    while (uniqueNumbers.size() < recordCount) {
        uniqueNumbers.insert(distr(gen));
    }

    auto it = uniqueNumbers.begin();
    for (int i = 0; i < recordCount; ++i) {
        Record rec;
        rec.cardNumber = *it++;
        strncpy(rec.name, names[i % names.size()].c_str(), sizeof(rec.name) - 1);
        strncpy(rec.address, addresses[i % addresses.size()].c_str(), sizeof(rec.address) - 1);
```

Рисунок 9 – Первая часть кода программы

```
        rec.name[sizeof(rec.name) - 1] = '\0';
        rec.address[sizeof(rec.address) - 1] = '\0';
        records.push_back(rec);
    }
}

// Функция записи данных в текстовый файл
void writeTextFile(const string& filename, const vector<Record>& records) {
    ofstream outFile(filename);
    if (!outFile) {
        cerr << "Ошибка при открытии файла " << filename << endl;
        return;
    }

    for (const auto& rec : records) {
        outFile << rec.cardNumber << "\t" << rec.name << "\t" << rec.address << endl;
    }
    outFile.close();
}

// Функция конвертации текстового файла в двоичный
void convertToBinary(const string& textFilename, const string& binaryFilename) {
    ifstream inFile(textFilename);
    ofstream outFile(binaryFilename, ios::binary);

    if (!inFile || !outFile) {
        cerr << "Ошибка при открытии файлов" << endl;
        return;
    }

    Record rec;
    while (inFile >> rec.cardNumber) {
        inFile.ignore();
        inFile.getline(rec.name, sizeof(rec.name), '\t');
        inFile.getline(rec.address, sizeof(rec.address));
        outFile.write(reinterpret_cast<char*>(&rec), sizeof(Record));
    }
```

Рисунок 10 – Вторая часть кода программы

```
// Основной код программы
int main() {
    const int recordCounts[] = {100, 1000, 10000}; // Различные размеры файлов
    vector<long long> searchTimes; // Времена выполнения поиска

    cout << "Размер файла | Время поиска (мкс)" << endl;
    cout << "-------------|--------------------" << endl;

    for (int count : recordCounts) {
        vector<Record> records;
        generateData(count, records); // Генерация данных
        writeTextFile("records.txt", records); // Запись в текстовый файл
        convertToBinary("records.txt", "records.bin"); // Конвертация в двоичный файл

        // Вызов функции для замера времени поиска
        long long duration = testSearchTime("records.bin", records[0].cardNumber); // Поиск первой записи
        searchTimes.push_back(duration);

        cout << count << "          | " << duration << endl;
    }

    return 0;
}
```

Рисунок 11 – Третья часть кода программы

2.5 Тестирование программы

Рассмотрим результат тестирования программы для 100 записей:

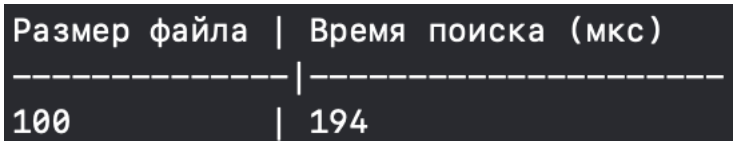

Рисунок 12 – Результат тестирования программы для 100 записей

2.6 Таблица с замерами времени поиска

Таблица 1. — Результаты замеров времени поиска

Размер файла	Время поиска в мкс
100	194
1000	136
10000	843

ЗАДАНИЕ №3

3.1 Постановка задания 3

Разработать программу для поиска записи в бинарном файле с применением алгоритма бинарного поиска и без использования дополнительной таблицы. Необходимо реализовать бинарный поиск записи по ключу (номер читательского билета) непосредственно в файле. Оценить эффективность алгоритма и провести замеры времени выполнения поиска на файлах с объемом 100, 1000 и 10 000 записей.

3.2 Описание алгоритма доступа к записи в файле посредством таблицы

В данной реализации поиск осуществляется непосредственно в бинарном файле, и при этом не используется дополнительная таблица. Программа открывает файл, определяет его размер и количество записей. Затем применяется бинарный поиск, в ходе которого файл читается только по мере необходимости. Запись из файла считывается при каждом шаге бинарного поиска, используя прямой доступ к данным по смещению.

3.3 Алгоритм поиска, определенный вариантом

Рассмотрим псевдокод функции поиска:

```
Function LinearSearch(FileName, key):
        FUNCTION binarySearchInFile(binaryFilename, searchKey)
            OPEN binary file binaryFilename FOR reading
            IF file could not be opened THEN
                PRINT error message
                RETURN false
            END IF

            GET file size and calculate recordCount
            SET low = 0
            SET high = recordCount - 1

            WHILE low <= high DO
                SET mid = (low + high) / 2
                SEEK file to position (mid * size of Record)
                READ record from file into foundRecord

                IF foundRecord.cardNumber == searchKey THEN
                    RETURN foundRecord
                ELSE IF foundRecord.cardNumber < searchKey THEN
                    SET low = mid + 1
```

```
            ELSE
               SET high = mid - 1
            END IF
         END WHILE

         RETURN not found
END FUNCTION
```

3.4 Код программы линейного поиска записи по ключу

Рассмотрим код программы:

```cpp
bool binarySearchInFile(const string& binaryFilename, int searchKey, Record& foundRecord) {
    ifstream inFile(binaryFilename, ios::binary);

    if (!inFile) {
        cerr << "Ошибка при открытии файла " << binaryFilename << endl;
        return false;
    }

    inFile.seekg(0, ios::end);
    size_t fileSize = inFile.tellg();
    size_t recordCount = fileSize / sizeof(Record);

    size_t low = 0;
    size_t high = recordCount - 1;

    while (low <= high) {
        size_t mid = (low + high) / 2;
        inFile.seekg(mid * sizeof(Record));
        inFile.read(reinterpret_cast<char*>(&foundRecord), sizeof(Record));

        if (foundRecord.cardNumber == searchKey) {
            return true;
        } else if (foundRecord.cardNumber < searchKey) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }

    return false;
}
```

Рисунок 12 – Функция бинарного поиска

3.5 Таблица с замерами времени поиска

Таблица 2. — Результаты замеров времени поиска

Размер файла	Время поиска в мкс
100	283
1000	301
10000	982

ВЫВОДЫ

В ходе работы с бинарным поиском мы узнали о его эффективности, особенно в сравнении с линейным поиском. Этот алгоритм позволяет значительно сократить время поиска в отсортированных данных благодаря своей логарифмической сложности O(log n). Мы изучили принцип его работы, который заключается в последовательном делении массива пополам, что помогает отсеять неинтересующие части и быстрее находить искомый элемент. Также мы осознали, что бинарный поиск можно применять только к отсортированным массивам, что требует предварительной сортировки данных.

Реализация алгоритма, как в итеративной, так и в рекурсивной форме, помогла нам лучше понять его внутреннюю структуру и особенности. Мы также столкнулись с типичными ошибками при реализации и научились их диагностировать, что улучшило наши навыки отладки кода. В итоге, работа с бинарным поиском не только улучшила наши алгоритмические способности, но и углубила понимание работы с данными и их поиском.

ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ

1. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.

2. Документация по языку C++ [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/cpp/cpp/ (дата обращения 08.09.2024).

3. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 04.09.2024)