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 ИКБО-41-23 Гольд Д.В.

Принял старший преподаватель Рысин.М.Л.

Москва 2024

СОДЕРЖАНИЕ

ЦЕЛЬ

Получить практический опыт по применению алгоритмов поиска в таблицах данных.

ЗАДАНИЕ

Разработать программу поиска записей с заданным ключом в двоичном файле с применением различных алгоритмов.

ОТЧЕТ ПО ЗАДАНИЮ 1

1.1 УСЛОВИЕ:

Задание 1 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.

1.2 Описание подхода к решению

Структурой записи файла является “Страховой полис: номер полиса, компания, фамилия владельца”

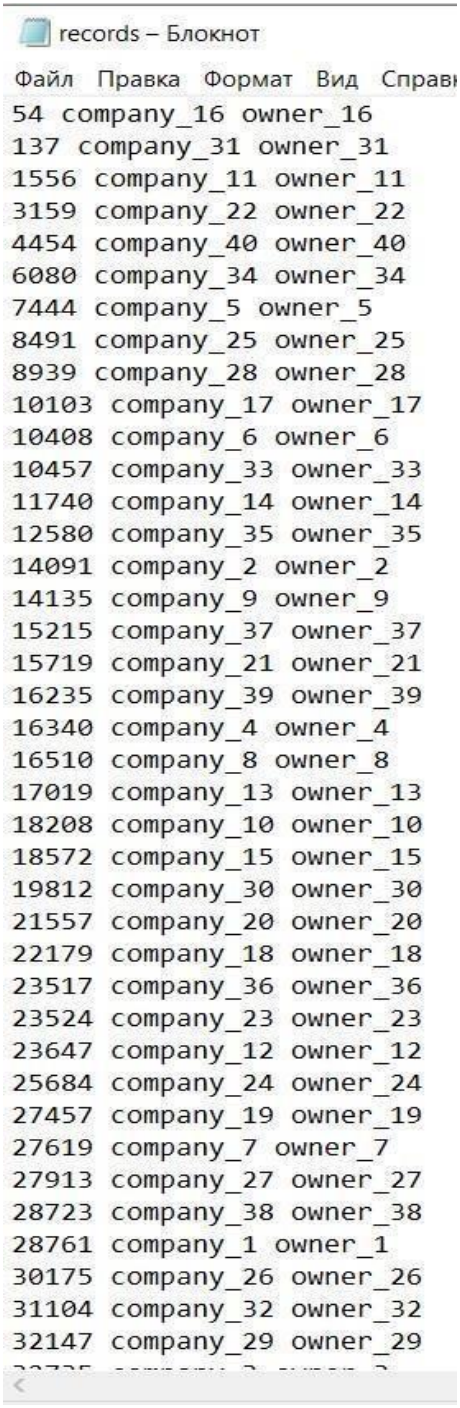
1.3 Код программы:

```cpp
class task_one {
public:
    struct infile_record_structure {
        int policy_number;
        string name_company;
        string owner_company_name;
    };

    //генерация случайных записей
    vector<infile_record_structure> generate_records(int count) {
        vector<infile_record_structure> records;
        srand(time(0));  //инициализация генератора случайных чисел

        for (int i = 0; i < count; ++i) {
            infile_record_structure record;

            //генерация случайного номера полиса
            record.policy_number = rand() % 1000000;

            //названия компании и владельца
            record.name_company = "company_" + to_string(i + 1);
            record.owner_company_name = "owner_" + to_string(i + 1);

            records.push_back(record);
        }

        //сортировка записей по номеру полиса (policy_number)
        sort(records.begin(), records.end(), [](const infile_record_structure& a, const infile_record_structure& b) {
            return a.policy_number < b.policy_number;
            });

        return records;
    }

    //запись текстового файла
    void write_to_text_file(const vector<infile_record_structure>& records, const string& filename) {
        ofstream outfile(filename);

        if (!outfile.is_open()) {
            cerr << "error opening file for writing" << endl;
            return;
        }

        for (const auto& record : records) {
            outfile << record.policy_number << " " << record.name_company << " " << record.owner_company_name << endl;
        }

```

```
// лаунчер
int main() {
    int user_choice;

    while (true) {
        SetConsoleTextAttribute(back_col, 0x0a);
        cout << "\nselect task:" << endl;
        cout << "1 - first task" << endl;
        cout << "2 - second task" << endl;
        cout << "3 - third task" << endl;
        cout << "4 - exit" << endl;
        cout << "enter u choice: ";
        SetConsoleTextAttribute(back_col, 0x07);
        cin >> user_choice;

        switch (user_choice) {
        case 1: {
            task_one object;
            int num_records;

            SetConsoleTextAttribute(back_col, 0x0a);
            cout << "enter the number of records to generate: ";
            SetConsoleTextAttribute(back_col, 0x07);
            cin >> num_records;

            //генерация записей
            vector<task_one::infile_record_structure> records = object.generate_records(num_records);

            //запись в текстовый файл
            object.write_to_text_file(records, "records.txt");
            cout << "text file 'records.txt' created" << endl;

            //запись в двоичный файл
            object.write_to_binary_file(records, "records.bin");
            cout << "binary file 'records.bin' created" << endl;
            break;
        }
```

1.4 Тестирование

```
enter u choice: 1
enter the number of records to generate: 40
text file 'records.txt' created
binary file 'records.bin' created
```



ОТЧЕТ ПО ЗАДАНИЮ 2

2.1 УСЛОВИЕ

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

2.2 Код программы

```
class task_two {
public:
    struct infile_record_structure {
        int policy_number;
        string name_company;
        string owner_company_name;
    };

public:
    //линейный поиск по ключу в бинарном файле
    infile_record_structure linear_search_in_binary_file(const string& filename, int key) {
        ifstream infile(filename, ios::binary);
        infile_record_structure record;

        if (!infile.is_open()) {
            cerr << "error opening binary file for reading" << endl;
            return record;
        }

        while (infile.read((char*)&record.policy_number, sizeof(record.policy_number))) {
            getline(infile, record.name_company, '\0');  //чтение строки до нуля
            getline(infile, record.owner_company_name, '\0');

            if (record.policy_number == key) {
                return record;//если нашли запись - возвращаем ее
            }
        }

        cerr << "record with key " << key << " not found!" << endl;
        return record;
    }
};
```

2.3 Тестирование

```
enter u choice: 2
enter the key (policy number) to search: 25728
record found: policy number: 25728, company: company_14, owner: owner_14
time: 1.0197 seconds
```

Таблица с практическим временем поиска в файле:

Количество записей в файле	Время поиска в секундах

100	1.0197
1000	1.5682
10000	2.4301

ОТЧЕТ ПО ЗАДАНИЮ 3

3.1 УСЛОВИЕ

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.

3.2 Код программы бинарного поиска

```
126  class task_three {
127  public:
128      struct IndexEntry {
129          int policy_number;
130          streampos file_offset;
131      };
132
133      //функция создания таблицы ключ-смещение из бинарного файла
134      vector<IndexEntry> create_index_table(const string& filename) {
135          vector<IndexEntry> index_table;
136          ifstream infile(filename, ios::binary);
137          task_one::infile_record_structure record;
138
139          if (!infile.is_open()) {
140              cerr << "error opening binary file for reading" << endl;
141              return index_table;
142          }
143
144          while (!infile.eof()) {
145              //получаем смещение текущей записи
146              streampos current_offset = infile.tellg();
147
148              //чтение policy_number
149              infile.read((char*)&record.policy_number, sizeof(record.policy_number));
150
151              if (infile.eof()) break;
152
153              //пропускаем строки (компанию и владельца)
154              infile.ignore((numeric_limits<streamsize>::max)(), '\0');
155              infile.ignore((numeric_limits<streamsize>::max)(), '\0');
156
157              //добавляем запись в таблицу
158              index_table.push_back({ record.policy_number, current_offset });
159          }
160
161          infile.close();
162
163          //сортировка таблицы индексов по policy_number
164          sort(index_table.begin(), index_table.end(), [](const IndexEntry& a, const IndexEntry& b) {
165              return a.policy_number < b.policy_number;
166              });
167
168          return index_table;
169      }
170
171      //функция для создания чисел Фибоначчи с использованием динамического программирования
172      vector<int> fibonacci_numbers(int n) {
173          vector<int> fib(n);
174          fib[0] = 0;
175          fib[1] = 1;
176
177          for (int i = 2; i < n; ++i) {
178              fib[i] = fib[i - 1] + fib[i - 2]; //сохраняем значения, чтобы не пересчитывать
```

```
170
171     //функция для создания чисел Фибоначчи с использованием динамического программирования
172     vector<int> fibonacci_numbers(int n) {
173         vector<int> fib(n);
174         fib[0] = 0;
175         fib[1] = 1;
176
177         for (int i = 2; i < n; ++i) {
178             fib[i] = fib[i - 1] + fib[i - 2]; //сохраняем значения, чтобы не пересчитывать
179         }
180
181         return fib;
182     }
183
184     //поиск по ключу в таблице индексов с использованием поиска Фибоначчи и динамического программирования
185     streampos fibonacci_search_in_index_table(const vector<IndexEntry>& index_table, int key) {
186         int n = index_table.size();
187
188         //получаем числа Фибоначчи
189         vector<int> fib = fibonacci_numbers(n);
190
191         //инициализация индексов
192         int fibM2 = fib[0];  // (m-2) = 0
193         int fibM1 = fib[1];  // (m-1) = 1
194         int fibM = fibM2 + fibM1; // m = fib(m)
195
196         //индекс для сравнения
197         int offset = -1;
198
199         //найти наибольшее число Фибоначчи меньшее или равное n
200         while (fibM < n) {
201             fibM2 = fibM1;
202             fibM1 = fibM;
203             fibM = fibM2 + fibM1;
204         }
205
206         //основной цикл поиска
207         while (fibM > 1) {
208             //индекс для сравнения
209             int i = min(offset + fibM2, n - 1);
210
211             //если ключ больше элемента по индексу i
212             if (index_table[i].policy_number < key) {
213                 fibM = fibM1;
214                 fibM1 = fibM2;
215                 fibM2 = fibM - fibM1;
216                 offset = i;
217             }
218             //еесли ключ меньше элемента по индексу i
219             else if (index_table[i].policy_number > key) {
220                 fibM = fibM2;
```

```
            //еесли ключ меньше элемента по индексу i
            else if (index_table[i].policy_number > key) {
                fibM = fibM2;
                fibM1 = fibM1 - fibM2;
                fibM2 = fibM - fibM1;
            }
            //элемент найден
            else return index_table[i].file_offset;
        }

        //сравнить последний элемент
        if (fibM1 && index_table[offset + 1].policy_number == key) {
            return index_table[offset + 1].file_offset;
        }

        //тсли элемент не найден
        cerr << "record with key " << key << " not found in the index table!" << endl;
        return -1;
    }

    //чтение записи по смещению из бинарного файла
    task_one::infile_record_structure read_record_by_offset(const string& filename, streampos offset) {
        ifstream infile(filename, ios::binary);
        task_one::infile_record_structure record;

        if (!infile.is_open()) {
            cerr << "error opening binary file for reading!" << endl;
            return record;
        }

        //переходим к нужному смещению
        infile.seekg(offset);

        //читаем запись
        infile.read((char*)&record.policy_number, sizeof(record.policy_number));
        getline(infile, record.name_company, '\0');
        getline(infile, record.owner_company_name, '\0');

        infile.close();
        return record;
    }
};
```

```cpp
// лаунчер
int main() {
    int user_choice;

    while (true) {
        SetConsoleTextAttribute(back_col, 0x0a);
        cout << "\nselect task:" << endl;
        cout << "1 - first task" << endl;
        cout << "2 - second task" << endl;
        cout << "3 - third task" << endl;
        cout << "4 - exit" << endl;
        cout << "enter u choice: ";
        SetConsoleTextAttribute(back_col, 0x07);
        cin >> user_choice;

        switch (user_choice) {
        case 1: { ... }
        case 2: { ... }
        case 3: {
            task_three object;

            //создание таблицы ключ-смещение
            vector<task_three::IndexEntry> index_table = object.create_index_table("records.bin");
            cout << "index table created" << endl;

            //поиск записи по ключу
            int key;
            SetConsoleTextAttribute(back_col, 0x0a);
            cout << "enter the key (policy number) to search in index table: ";
            SetConsoleTextAttribute(back_col, 0x07);
            cin >> key;

            streampos offset = object.search_in_index_table(index_table, key);

            //чтение записи по смещению
            if (offset != -1) {
                task_one::infile_record_structure result = object.read_record_by_offset("records.bin", offset);
                cout << "record found: policy number: " << result.policy_number
                    << ", company: " << result.name_company
                    << ", owner: " << result.owner_company_name << endl;
            }
            break;
        }
```

3.3 Тестирование

```
enter u choice: 3
index table created
enter the key (policy number) to search in index table: 25728
record found: policy number: 25728, company: company_14, owner: owner_14
time: 1.24388 seconds
```

Таблица с практическим временем поиска в файле:

Количество записей в файле	Время поиска в секундах

100	0.052
1000	0.0852
10000	1.24388

ВЫВОД

Получил практический опыт по применению алгоритмов поиска в таблицах данных.