



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.....	3
ЗАДАНИЕ	3
ОТЧЕТ ПО ЗАДАНИЮ 1.....	3
1.1 УСЛОВИЕ:	3
1.2 Описание подхода к решению	3
1.3 Код программы:.....	4
1.4 Тестирование	5
ОТЧЕТ ПО ЗАДАНИЮ 2.....	6
2.1 УСЛОВИЕ.....	6
2.2 Код программы.....	7
2.3 Тестирование	8
ОТЧЕТ ПО ЗАДАНИЮ 3	9
3.1 УСЛОВИЕ	9
3.2 Описание алгоритма создания таблицы.....	9
3.3 Код программы создания таблицы	10
ВЫВОД	14

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:

```
18 class task_one {
19 public:
20     struct infile_record_structure {
21         int policy_number;
22         string name_company;
23         string owner_company_name;
24     };
25
26     //генерация случайных записей
27     vector<infile_record_structure> generate_records(int count) {
28         vector<infile_record_structure> records;
29         srand(time(0)); //инициализация генератора случайных чисел
30
31         for (int i = 0; i < count; ++i) {
32             infile_record_structure record;
33
34             //генерация случайного номера полиса
35             record.policy_number = rand() % 1000000;
36
37             //названия компании и владельца
38             record.name_company = "company_" + to_string(i + 1);
39             record.owner_company_name = "owner_" + to_string(i + 1);
40
41             records.push_back(record);
42         }
43
44         //сортировка записей по номеру полиса (policy_number)
45         sort(records.begin(), records.end(), [](const infile_record_structure& a, const infile_record_structure& b) {
46             return a.policy_number < b.policy_number;
47         });
48
49         return records;
50     }
51
52     //запись текстового файла
53     void write_to_text_file(const vector<infile_record_structure>& records, const string& filename) {
54         ofstream outfile(filename);
55
56         if (!outfile.is_open()) {
57             cerr << "error opening file for writing" << endl;
58             return;
59         }
60
61         for (const auto& record : records) {
62             outfile << record.policy_number << " " << record.name_company << " " << record.owner_company_name << endl;
63         }
64     }
```

```

203 // лаунчер
204 int main() {
205     int user_choice;
206
207     while (true) {
208         SetConsoleTextAttribute(back_col, 0x0a);
209         cout << "\nselect task:" << endl;
210         cout << "1 - first task" << endl;
211         cout << "2 - second task" << endl;
212         cout << "3 - third task" << endl;
213         cout << "4 - exit" << endl;
214         cout << "enter u choice: ";
215         SetConsoleTextAttribute(back_col, 0x07);
216         cin >> user_choice;
217
218         switch (user_choice) {
219             case 1: {
220                 task_one object;
221                 int num_records;
222
223                 SetConsoleTextAttribute(back_col, 0x0a);
224                 cout << "enter the number of records to generate: ";
225                 SetConsoleTextAttribute(back_col, 0x07);
226                 cin >> num_records;
227
228                 //генерация записей
229                 vector<task_one::infile_record_structure> records = object.generate_records(num_records);
230
231                 //запись в текстовый файл
232                 object.write_to_text_file(records, "records.txt");
233                 cout << "text file 'records.txt' created" << endl;
234
235                 //запись в двоичный файл
236                 object.write_to_binary_file(records, "records.bin");
237                 cout << "binary file 'records.bin' created" << endl;
238                 break;
239             }

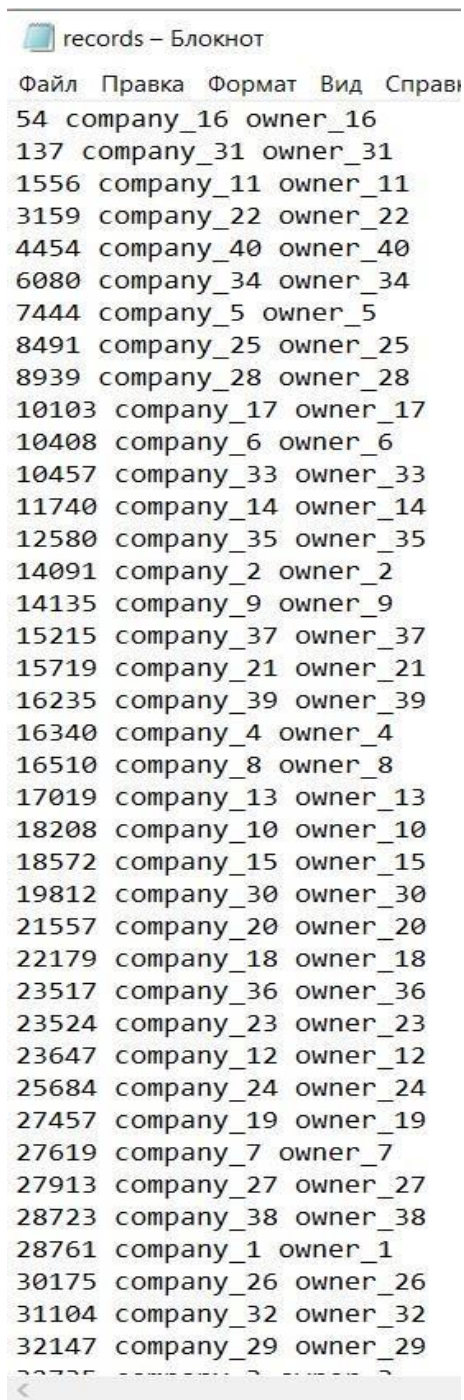
```

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

```
89 class task_two {
90 public:
91     struct infile_record_structure {
92         int policy_number;
93         string name_company;
94         string owner_company_name;
95     };
96
97 public:
98     //линейный поиск по ключу в бинарном файле
99     infile_record_structure linear_search_in_binary_file(const string& filename, int key) {
100         ifstream infile(filename, ios::binary);
101         infile_record_structure record;
102
103         if (!infile.is_open()) {
104             cerr << "error opening binary file for reading" << endl;
105             return record;
106         }
107
108         while (infile.read((char*)&record.policy_number, sizeof(record.policy_number))) {
109             getline(infile, record.name_company, '\0'); //чтение строки до нуля
110             getline(infile, record.owner_company_name, '\0');
111
112             if (record.policy_number == key) {
113                 return record; //если нашли запись - возвращаем ее
114             }
115         }
116
117         cerr << "record with key " << key << " not found!" << endl;
118         return record;
119     }
120 };
121
```



```

121     return record;
122 }
123
124 };
125
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

```



```

224     }
225
226     //если элемент не найден
227     cerr << "record with key " << key << " not found in the index table!" << endl;
228     return -1;
229 }
230
231 //чтение записи по смещению из бинарного файла
232 task_one::infile_record_structure read_record_by_offset(const string& filename, streampos offset) {
233     ifstream infile(filename, ios::binary);
234     task_one::infile_record_structure record;
235
236     if (!infile.is_open()) {
237         cerr << "error opening binary file for reading!" << endl;
238         return record;
239     }
240
241     //переходим к нужному смещению
242     infile.seekg(offset);
243
244     //читаем запись
245     infile.read((char*)&record.policy_number, sizeof(record.policy_number));
246     getline(infile, record.name_company, '\0');
247     getline(infile, record.owner_company_name, '\0');
248
249     infile.close();
250     return record;
251 }
252 };
253
254
255 //поиск по ключу в таблице индексов с использованием поиска Фибоначчи
256 streampos fibonacci_search_in_index_table(const vector<IndexEntry>& index_table, int key) {
257     int n = index_table.size();
258     int fibM2 = 0; // (m-2)
259     int fibM1 = 1; // (m-1)
260     int fibM = fibM2 + fibM1;
261
262     //найти Fibonacci number < n
263     while (fibM < n) {
264         fibM2 = fibM1;
265         fibM1 = fibM;
266         fibM = fibM2 + fibM1;
267     }
268
269     //индекс -который будет использоваться для сравнения
270     int offset = -1;
271
272     while (fibM > 1) {
273         //индекс для сравнения
274         int i = min(offset + fibM2, n - 1);
275
276         //если key больше - чем элемент - по индексу i
277         if (index_table[i].policy_number < key) {
278             fibM = fibM1;
279             fibM1 = fibM2;
280             fibM2 = fibM - fibM1;
281             offset = i; //переход к следующей части
282         }
283         //если key меньше - чем элемент - по индексу i
284         else if (index_table[i].policy_number > key) {
285             fibM = fibM2;
286             fibM1 = fibM1 - fibM2;
287             fibM2 = fibM - fibM1;
288         }
289         //элемент найден
290         else return index_table[i].file_offset;
291     }
292
293     //сравнить последний элемент
294     if (fibM1 && index_table[offset + 1].policy_number == key) {
295         return index_table[offset + 1].file_offset;
296     }
297
298     //если элемент не найден

```

```

179     cerr << "Record with key " << key << " not found in the index table!" << endl;
180     return -1;
181 }
182
183 //чтение записи по смещению из бинарного файла
184 task_one::infile_record_structure read_record_by_offset(const string& filename, streampos offset) {
185     ifstream infile(filename, ios::binary);
186     task_one::infile_record_structure record;
187
188     if (!infile.is_open()) {
189         cerr << "error opening binary file for reading!" << endl;
190         return record;
191     }
192
193     //переходим к нужному смещению
194     infile.seekg(offset);
195
196     //читаем запись
197     infile.read((char*)&record.policy_number, sizeof(record.policy_number));
198     getline(infile, record.name_company, '\0');
199     getline(infile, record.owner_company_name, '\0');
200
201     infile.close();
202     return record;
203 }
204 };
205
206
207 // лаунчер
208 int main() {
209     int user_choice;
210
211     while (true) {

```

```

208 int main() {
209     int user_choice;
210
211     while (true) {
212         SetConsoleTextAttribute(back_col, 0x0a);
213         cout << "\nselect task:" << endl;
214         cout << "1 - first task" << endl;
215         cout << "2 - second task" << endl;
216         cout << "3 - third task" << endl;
217         cout << "4 - exit" << endl;
218         cout << "enter u choice: ";
219         SetConsoleTextAttribute(back_col, 0x07);
220         cin >> user_choice;
221
222         switch (user_choice) {
223             case 1: { ... }
224             case 2: {
225                 auto start = std::chrono::high_resolution_clock::now();
226
227                 task_two object;
228                 int key;
229
230                 SetConsoleTextAttribute(back_col, 0x0a);
231                 cout << "enter the key (policy number) to search: ";
232                 SetConsoleTextAttribute(back_col, 0x07);
233                 cin >> key;
234
235                 task_two::infile_record_structure result = object.linear_search_in_binary_file("records.bin", key);
236
237                 if (result.policy_number != 0) { //если нашли запись
238                     cout << "record found: policy number: " << result.policy_number
239                         << ", company: " << result.name_company
240                         << ", owner: " << result.owner_company_name << endl;
241                     auto end = chrono::high_resolution_clock::now();
242                     chrono::duration<double> duration = end - start;
243                     cout << "time: " << duration.count() << " seconds\n";
244                 }
245                 break;
246             }
247             default: {
248                 break;
249             }
250         }
251     }
252 }

```

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 предвычисления чисел Фибоначчи
172     vector<int> generate_fibonacci(int n) {
173         vector<int> fib(n + 2, 0); // создаем массив для хранения чисел Фибоначчи
174         fib[0] = 0;
```

```

171 //функция для предвычисления чисел Фибоначчи
172 vector<int> generate_fibonacci(int n) {
173     vector<int> fib(n + 2, 0); // создаем массив для хранения чисел Фибоначчи
174     fib[0] = 0;
175     fib[1] = 1;
176     for (int i = 2; i <= n; ++i) {
177         fib[i] = fib[i - 1] + fib[i - 2];
178     }
179     return fib;
180 }
181
182 //поиск по ключу в таблице индексов с использованием поиска Фибоначчи
183 streampos fibonacci_search_in_index_table(const vector<IndexEntry>& index_table, int key) {
184     int n = index_table.size();
185
186     //генерация всех чисел Фибоначчи до размера массива
187     vector<int> fib = generate_fibonacci(n);
188
189     //инициализация индексов
190     int fibM2 = 0; // (m-2)
191     int fibM1 = 1; // (m-1)
192     int fibM = fibM2 + fibM1; // m-е число Фибоначчи
193
194     //найдем наибольшее число Фибоначчи - меньшее или равное n
195     while (fibM < n) {
196         fibM2 = fibM1;
197         fibM1 = fibM;
198         fibM = fibM1 + fibM2;
199     }
200
201     //инициализация оффсета
202     int offset = -1;
203
204     //пока больше одного элемента
205     while (fibM > 1) {
206         int i = min(offset + fibM2, n - 1);
207
208         if (index_table[i].policy_number < key) {
209             fibM = fibM1;
210             fibM1 = fibM2;
211             fibM2 = fibM - fibM1;
212             offset = i;
213         }
214         else if (index_table[i].policy_number > key) {
215             fibM = fibM2;
216             fibM1 = fibM1 - fibM2;
217             fibM2 = fibM - fibM1;
218         }

```



```

214         else if (index_table[i].policy_number > key) {
215             fibM = fibM2;
216             fibM1 = fibM1 - fibM2;
217             fibM2 = fibM - fibM1;
218         }
219         else {
220             return index_table[i].file_offset;
221         }
222     }
223
224     //проверяем последний элемент
225     if (fibM1 && index_table[offset + 1].policy_number == key) {
226         return index_table[offset + 1].file_offset;
227     }
228
229     cerr << "record with key " << key << " not found in the index table!" << endl;
230     return -1;
231 }
232
233 //чтение записи по смещению из бинарного файла
234 task_one::infile_record_structure read_record_by_offset(const string& filename, streampos offset) {
235     ifstream infile(filename, ios::binary);
236     task_one::infile_record_structure record;
237
238     if (!infile.is_open()) {
239         cerr << "error opening binary file for reading!" << endl;
240         return record;
241     }
242
243     //переходим к нужному смещению
244     infile.seekg(offset);
245
246     //читаем запись
247     infile.read((char*)&record.policy_number, sizeof(record.policy_number));
248     getline(infile, record.name_company, '\0');
249     getline(infile, record.owner_company_name, '\0');
250
251     infile.close();
252     return record;
253 }
254 };
255

```



```

203 // лаунчер
204 int main() {
205     int user_choice;
206
207     while (true) {
208         SetConsoleTextAttribute(back_col, 0x0a);
209         cout << "\nselect task:" << endl;
210         cout << "1 - first task" << endl;
211         cout << "2 - second task" << endl;
212         cout << "3 - third task" << endl;
213         cout << "4 - exit" << endl;
214         cout << "enter u choice: ";
215         SetConsoleTextAttribute(back_col, 0x07);
216         cin >> user_choice;
217
218         switch (user_choice) {
219             case 1: { ... }
240             case 2: { ... }
258             case 3: {
259                 task_three object;
260
261                 //создание таблицы ключ-смещение
262                 vector<task_three::IndexEntry> index_table = object.create_index_table("records.bin");
263                 cout << "index table created" << endl;
264
265                 //поиск записи по ключу
266                 int key;
267                 SetConsoleTextAttribute(back_col, 0x0a);
268                 cout << "enter the key (policy number) to search in index table: ";
269                 SetConsoleTextAttribute(back_col, 0x07);
270                 cin >> key;
271
272                 streampos offset = object.search_in_index_table(index_table, key);
273
274                 //чтение записи по смещению
275                 if (offset != -1) {
276                     task_one::infile_record_structure result = object.read_record_by_offset("records.bin", offset);
277                     cout << "record found: policy number: " << result.policy_number
278                         << ", company: " << result.name_company
279                         << ", owner: " << result.owner_company_name << endl;
280                 }
281                 break;
282             }
283         }
284     }
285 }

```

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.