

Отчёт по выполнению практического задания № 5.2
Тема:
«Работа с данными из файла»

Группа: ИКБО-10-23

Москва 2024

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.

Вариант:

Номер: 6

Алгоритм поиска:

Бинарный однородный без использования дополнительной таблицы

Структура записи файла (ключ – подчеркнутое поле):

Товар: название, код – шестизначное число

1.1 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.

1.2 Ход решения

По рекомендации, оставленной в методических указаниях к практической работе, было решено сначала генерировать текстовый файл, а потом преобразовывать его в двоичный.

Заранее был создан файл с сотней случайных наименований товаров. Этот файл считывается в массив, из которого выбирается случайное наименование, которому в пару ставится случайное 6-значное число — код товара. Эта информация построчно записывается в текстовый файл.

После сохранения текстового файла он открывается для чтения и данные построчно считываются, после чего создаётся указатель на структуру `product`, в которую сохраняются данные товара. А уже структура записывается в двоичный файл. Так как она должна иметь фиксированный размер для корректной записи, данные в «сыром» виде сохранять нельзя, из-за чего используется структура.

Структура содержит одно поле типа `int`, а также статический массив типа `char` длиной в 50 символов, поэтому объём структуры: $4 \text{ байта} + 50 * 1 \text{ байт} = 54 \text{ байта}$.

Так как структуры имеют фиксированный размер, становится возможен прямой доступ к записям, сравнимый по сложности с индексацией в массиве. Для прямого доступа к n -й структуре необходимо сместиться на $n * s$ байт, где s — размер в байтах одной структуры, после чего считать s байт, которые и будут являться нужной записью.

1.3 Исходный код программ

На рисунках 1-3 представлен исходный коды программ 1.а и 1.б, а также код листинга 1.

```
12 struct product
13 {
14     int code;
15     char name[50];
16 };
```

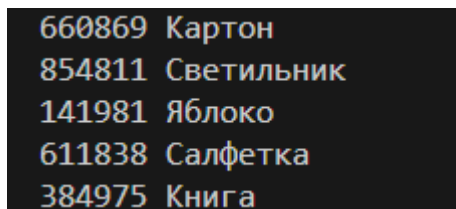
Рисунок 1 — Структура product

```
34 void generate_file(string fname, int cnt)
35 {
36     random_device rd;
37     mt19937 gen(rd());
38
39     ifstream product_names;
40     product_names.open("product_names.txt");
41     vector<string> names;
42     string p_name;
43     while (!product_names.eof())
44     {
45         getline(product_names, p_name);
46         names.push_back(p_name);
47     }
48     product_names.close();
49
50     fstream temp;
51     temp.open("temp.txt", ios::out | ios::trunc);
52
53     for (int i = 0; i < cnt; ++i)
54     {
55         temp << (100000 + rd() % 899999) << " " << names[rd() % names.size()] << '\n';
56     }
57     temp.close();
58     temp.open("temp.txt", ios::in);
59
60     fstream file;
61     file.open(fname, ios::binary | ios::out | ios::trunc);
62
63     product *x = new product();
64
65     while (temp >> x->code >> x->name)
66     {
67         file.write((char *)x, sizeof(product));
68     }
69
70     // sort(for_sort.begin(), for_sort.end(), compare_products);
71     file.close();
72     temp.close();
73 }
```

Рисунок 2 - Исходный код функции generate_file

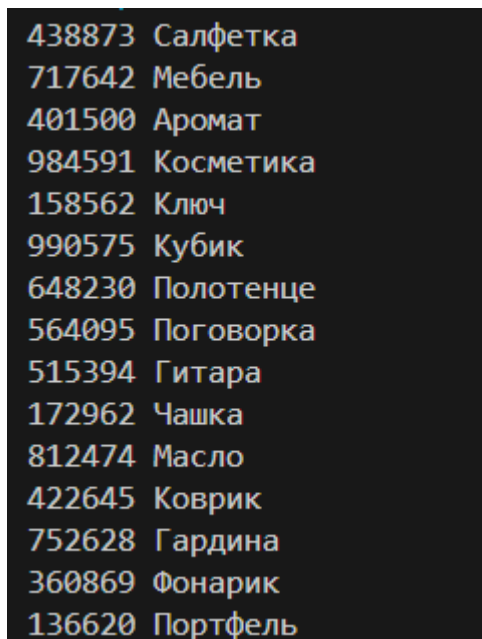
1.4 Результаты тестирования

На рисунках 3-4 представлены результаты тестирования генерации двоичных файлов для 5 и 15 записей соответственно.



```
660869 Картон
854811 Светильник
141981 Яблоко
611838 Салфетка
384975 Книга
```

Рисунок 3 — Результат тестирования для 5 записей



```
438873 Салфетка
717642 Мебель
401500 Аромат
984591 Косметика
158562 Ключ
990575 Кубик
648230 Полотенце
564095 Поговорка
515394 Гитара
172962 Чашка
812474 Масло
422645 Коврик
752628 Гардина
360869 Фонарик
136620 Портфель
```

Рисунок 4 — Результат тестирования для 15 записей

2.1 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.

2.2 Описание алгоритма

Линейный поиск очень простой алгоритм, из-за чего его сложность является линейной: $O(n)$. Необходимо поочередно считывать все данные, сохраняя их в нужную структуру, и у этой структуры проверять совпадение кода с искомым. При совпадении необходимо вывести номер текущей записи, являющийся смещением.

2.3 Исходный код программ

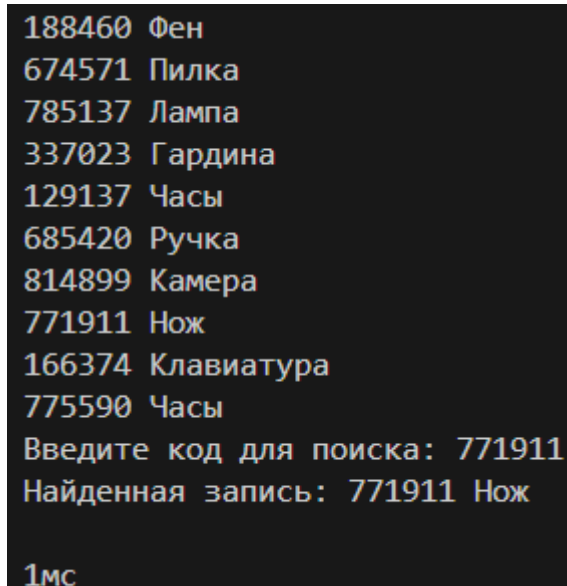
На рисунке 5 представлен исходный код программы линейного поиска.

```
87 void linear_search(string fname, int code)
88 {
89     fstream file;
90     file.open(fname, ios::binary | ios::in);
91     product *x = new product();
92     int i = 0;
93     while (file.read((char *)x, sizeof(product)))
94     {
95         if (x->code == code)
96         {
97             file.close();
98             return i;
99         }
100         ++i;
101     }
102     file.close();
103     return -1;
104 }
```

Рисунок 5 — Исходный код функции linear_search

2.4 Результаты тестирования

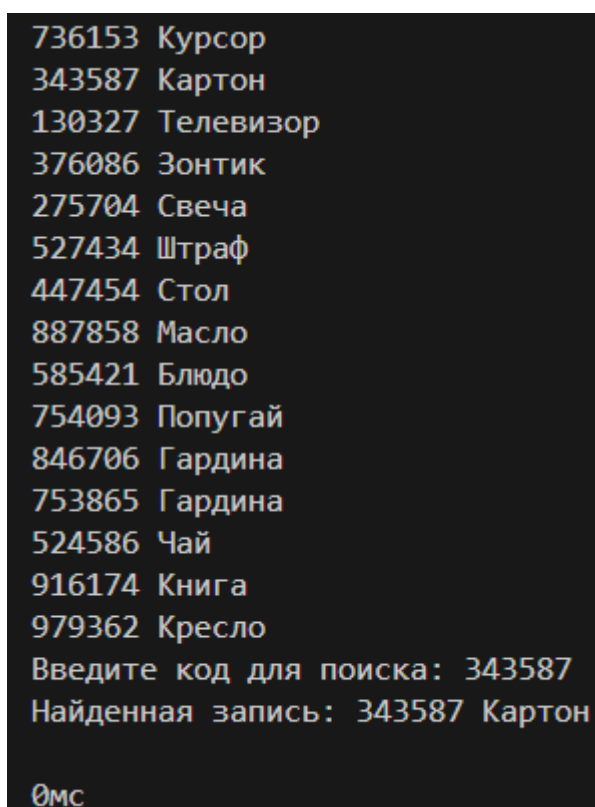
На рисунках 6-7 представлены результаты тестирования алгоритма линейного поиска, а в таблице 1 представлены результаты тестирования с замераами времени.



```
188460 Фен
674571 Пилка
785137 Лампа
337023 Гардина
129137 Часы
685420 Ручка
814899 Камера
771911 Нож
166374 Клавиатура
775590 Часы
Введите код для поиска: 771911
Найденная запись: 771911 Нож

1мс
```

Рисунок 6 — Результат тестирования



```
736153 Курсор
343587 Картон
130327 Телевизор
376086 Зонтик
275704 Свеча
527434 Штраф
447454 Стол
887858 Масло
585421 Блюдо
754093 Попугай
846706 Гардина
753865 Гардина
524586 Чай
916174 Книга
979362 Кресло
Введите код для поиска: 343587
Найденная запись: 343587 Картон

0мс
```

Рисунок 7 — Результат тестирования

Таблица 1 — Результаты тестирования линейного поиска

N	T, мс
100	0
1 000	0
10 000	1
100 000	4

3.1 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 структуру 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 Описание алгоритма

Для оптимизации была создана структура `table`, у которой два поля типа `int`: код товара и ссылка на запись в файле с соответствующим кодом. Ссылка на запись представляет собой индекс нужной записи в файле. Благодаря смещению «курсора» в файле на размер структуры, умноженный на количество структур для пропуска, можно получить прямой доступ к нужной записи в файле.

В начале происходит считывание всех данных из файла. Каждая запись сохраняется в структуру `table` в массиве структур. После считывания происходит сортировка по ключам и выполняется однородный бинарный поиск.

Однородный (равномерный) бинарный поиск является модификацией обычного бинарного поиска. Эта модификация заключается в следующем. Вместо трех указателей – l (нижняя граница интервала), h (верхняя граница интервала) и m (середина интервала) – используется только два: текущая

позиция m и величина его изменения δ . После каждого сравнения, не давшего равенства, можно установить $m \leftarrow m \pm \delta$ и $\delta \leftarrow \delta / 2$.

3.3 Исходный код программы

На рисунке 8 представлен исходный код алгоритма бинарного поиска. На рисунке 9 представлен алгоритм считывания записи из файла при помощи прямого доступа.

```
106 int bin_search(char *fname, int code)
107 {
108     fstream rd;
109     rd.open(fname, ios::in | ios::binary);
110
111     vector<table *> tb;
112
113     int i = 0;
114     product *x = new product();
115     while (rd.read((char *)x, sizeof(product)))
116     {
117         tb.emplace_back(new table(x->code, i));
118         ++i;
119     }
120     rd.close();
121
122     sort(tb.begin(), tb.end(), compare_tables);
123
124
125     if (tb.size() % 2 != 0)
126     {
127         tb.insert(tb.begin(), new table(-1, 0));
128     }
129
130     tb.insert(tb.begin(), new table(-1, 0));
131
132     int m = ceil(tb.size() / 2.0);
133     int q = floor(tb.size() / 2.0);
134
135     while (q > 0)
136     {
137         if (tb[m]->code == code)
138         {
139             break;
140         }
141
142         if (tb[m]->code < code)
143         {
144             m += ceil(q / 2.0);
145         }
146         else
147         {
148             m -= ceil(q / 2.0);
149         }
150         q = floor(q / 2.0);
151     }
152
153     if (tb[m]->code == code)
154     {
155         return tb[m]->offset;
156     }
157     return -1;
158 }
```

Рисунок 8 — Исходный код однородного бинарного поиска

```

160  product *read_record(char *fname, int offset)
161  {
162      product *x = new product();
163      FILE *file = fopen(fname, "rb");
164      fseek(file, sizeof(product) * offset, SEEK_SET);
165      fread(x, sizeof(product), 1, file);
166      fclose(file);
167      return x;
168  }

```

Рисунок 9 — Исходный код функции считывания записи по ссылке

2.4 Результаты тестирования

На рисунке 10-11 представлены результаты тестирования алгоритма поиска. В таблице 2 представлены результаты тестирования программы с учётом времени выполнения.

```

386107 Холодильник
177044 Курсор
598761 Скейтборд
955842 Тетрадь
572125 Рука
Введите код для поиска: 955842
Найденная запись: 955842 Тетрадь

0мс

```

Рисунок 10 — Результат тестирования для 5 записей

```

893748 Палитра
290177 Рукавичка
752492 Айс
707563 Товар
413842 Годинник
691216 Диван
551857 Тапочки
641223 Полотенце
326749 Картридж
461034 Часы
938226 Тапочки
467147 Аромат
289954 Блюдо
846287 Календарь
739223 Палитра
Введите код для поиска: 467147
Найденная запись: 467147 Аромат

0мс

```

Рисунок 16 — Результат сортировки последовательности

Таблица 2 — Результаты тестирования бинарного поиска

N	T, мс
100	1
1 000	1
10 000	11
100 000	414

3.1 Выводы

В ходе решения поставленных задач был получен практический опыт по применению алгоритмов поиска в таблицах данных.

Поиск в таблицах данных обладает преимуществом оптимизации и удобства доступа, так как структуры данных гораздо удобнее и быстрее сортировать, с ними удобнее и безопаснее работать, чем напрямую с файлами. Благодаря использованию же ссылок на записи в файле оказывается возможно выполнять поиск по таблице данных, после чего получать строку уже непосредственно из файла.