



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания 5.2

Тема: «Работа с данными из файла»

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент Фамилия И. О.
Группа АААА-00-00

Москва 2024

СОДЕРЖАНИЕ

1 ВВЕДЕНИЕ.....	3
1.1 Цель работы.....	3
1.2 Постановка задач.....	3
2 ЗАДАНИЕ 1.....	4
2.1 Постановка задачи.....	4
2.2 Анализ решения.....	4
2.3 Код программы.....	4
2.4 Результаты тестирования.....	6
3 ЗАДАНИЕ 2.....	7
3.1 Постановка задачи.....	7
3.2 Анализ решения.....	7
3.3 Код программы.....	7
3.4 Результаты тестирования.....	8
4 ЗАДАНИЕ 3.....	9
4.1 Постановка задачи.....	9
4.2 Анализ решения.....	9
4.3 Код программы.....	10
4.4 Результаты тестирования.....	11
5 ВЫВОД.....	12
6 ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ.....	13

1 ВВЕДЕНИЕ

1.1 Цель работы

Поучить практический опыт по применению алгоритмов поиска в таблицах данных.

1.2 Постановка задач

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

2 ЗАДАНИЕ 1

2.1 Постановка 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.

Индивидуальный вариант: Владелец телефона: номер телефона – последовательность символов, адрес

2.2 Анализ решения

Сгенерируем файл с записями в соответствии с индивидуальным вариантом в текстовом формате. Для генерации уникальных ключей добавим проверку на уникальность при помощи set. Преобразуем файл в бинарный формат путём преобразования числовых ключей в бинарный формат. Адрес запишем посимвольно напрямую в файл.

Однако для чтения и записи бинарного файла необходимо соблюдать постоянность битовых значений записей. Поэтому выделим минимально необходимые битовые размеры записей элементов. Для номера хватит 11 бит, для адреса выделим 50 бит. На одну запись выйдет $11 * 50 / 8 = 68.75$ байт. Итого на файл размером 100 записей понадобится $11 * 50 * 100 / 8 = 6875$ байт.

Код генерации файла представлен в листинге 1.

Результаты тестирования представлены на рис 1 и 2.

2.3 Код программы

Листинг 1 - Функция генерация бинарного файла

```
string streets[7] {"Ул. Ленина", "Пр. Вернадского", "Ул.
Покрышкина", "Ул. Коротышкина", "Пр. Нахимовский", "Ул.
Прямая", "Ул. Кривая"};
#define NUM_SIZE 11
#define ADDR_SIZE 50
long long int key = 88005553535;
string key_addr = "Пр. Вернадского Дом №86";

void generate_file(int file_size, string file_name) {
```

```

// Генерация txt файла
ofstream file(file_name + ".txt");
set<long long int> uniqs;
uniqs.insert(key);

for (int i = 0; i < file_size - 1; i++)
{
    // Уникальная генерация
    long long int unum = rnd();
    while(!uniqs.insert(unum).second) {
        unum = rnd();
    }
    string num = to_string(unum);
    string addr = streets[rnd2()] + " Дом №" +
to_string(rnd3());

    num.insert(0, NUM_SIZE - num.length() - 1, '0');
    num = "8" + num;
    addr.insert(addr.end(), 50 - addr.size(), '9');

    file << num << ";" << addr << endl;
}
key_addr.insert(key_addr.end(), 50 - key_addr.size(), ' ');
file << key << ";" << key_addr;
file.close();

ifstream txtfile(file_name + ".txt");
ofstream binfile(file_name + ".bin", ios::binary |
ios::out);
string line;

while(getline(txtfile, line)) {
    istringstream ss(line);
    string num, addr;
    long long int lnum;

    getline(ss, num, ';');
    getline(ss, addr);

    lnum = stoll(num);
    char addr_buf[ADDR_SIZE];
    strcpy(addr_buf, addr.c_str());

    binfile.write(reinterpret_cast<char *>(&lnum),
sizeof(lnum));
    binfile.write(addr_buf, ADDR_SIZE);
}
}

```

2.4 Результаты тестирования

```
output > data100.txt
1 83646826415;Пр. Верндадского Дом №189
2 88786909094;Пр. Нахимовский Дом №105
3 83441708803;Ул. Прямая Дом №99
4 83689530703;Ул. Покрышкина Дом №100
5 84800912496;Ул. Прямая Дом №203
6 86396039279;Ул. Кривая Дом №83
7 83224991648;Пр. Верндадского Дом №174
8 89239867275;Пр. Нахимовский Дом №50
9 89280229960;Ул. Коротышкина Дом №218
10 82378052367;Пр. Нахимовский Дом №5
11 89925649457;Ул. Кривая Дом №159
12 89964148741;Пр. Верндадского Дом №47
13 83714393443;Ул. Ленина Дом №40
14 80230714916;Ул. Ленина Дом №231
15 89120334685;Ул. Ленина Дом №91
16 82902091464;Ул. Кривая Дом №237
17 85677781782;Ул. Коротышкина Дом №37
18 84380314542;Ул. Коротышкина Дом №43
```

Рисунок 1 - Сгенерированный текстовый файл

```
output > data100.bin
1 0G0y5555Пр. Верндадского Дом №189 0005555Пр. Нахимовский Дом №105
2 W5555Пр. Нахимовский Дом №131 &005555Ул. Прямая Дом №5
3 p05555Ул. Ленина Дом №136 0005555Ул. Покрышкина Дом №190
```

Рисунок 2 - Преобразованный бинарный файл

3 ЗАДАНИЕ 2

3.1 Постановка 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 Анализ решения

Будем побитово считывать из бинарного файла количество битов ранее заданного размера. Преобразуем битовый формат ключа в число и будем сравнивать с поисковым элементом. При совпадении, остановим поиск и вернём результат.

Код функции линейного поиска представлен в листинге 2.

Результаты тестирования представлены на рис 3. Таблица результатов представлена в таблице 1.

3.3 Код программы

Листинг 2 - Функция линейного поиска по бинарному файлу

```
#define NUM_SIZE 11
#define ADDR_SIZE 50

string linear_search(string file_name, long long int key){
    ifstream input(file_name + ".bin", ios::binary);

    char addr_buf[ADDR_SIZE + 1] = {0};
    long long int lnum;
    string result = "";

    auto start = high_resolution_clock::now();
    while(input.read(reinterpret_cast<char*>(&lnum),
sizeof(lnum))) {
        input.read(addr_buf, ADDR_SIZE);

        if(lnum == key) {
            result = to_string(lnum) + " " + string(addr_buf);
            break;
        }
    }
```

```

    }
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop -
start).count();
    cout << file_name << ": Время: " << duration << " мкс." <<
endl;
    return result;
}

```

3.4 Результаты тестирования

```

Linear search
data100: Время: 13 мкс.
data1000: Время: 48 мкс.
data10000: Время: 344 мкс.

```

Рисунок 3 - Результаты
тестирования линейного поиска

Таблица 1 — Сравнение тестирования линейного поиска

Линейный поиск			
Запуск	1	2	3
100 записей	13 мс	10 мс	11 мс
1000 записей	48 мс	32 мс	33 мс
10000 записей	344 мс	293 мс	336 мс

4 ЗАДАНИЕ 3

4.1 Постановка 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.

Индивидуальный вариант — Бинарный однородный с использованием таблицы смещений.

4.2 Анализ решения

Создадим таблицу смещений в виде вектора пар ключ — значение: номер — смещение. Отсортируем её по ключам используя внутреннюю функцию сортировки `sort`. Далее используя однородный бинарный поиск по ключам найдём поисковое значение. Однородный бинарный поиск использует смещение вместо флагов, как в классическом бинарном поиске.

Код функции бинарного поиска представлен в листинге 3.

Результаты тестирования представлены на рис 4. Таблица результатов представлена в таблице 5.

4.3 Код программы

Листинг 3 - Функция однородного бинарного поиска по бинарному файлу

```
string binary_offset_search(string file_name, long long int
key)
{
    ifstream input(file_name + ".bin", ios::binary);

    char addr_buf[ADDR_SIZE + 1] = {0};
    long long int lnum;

    // Initializing offset table
    vector<pair<long long int, int>> offset_table;
    int offset = sizeof(long long int) + ADDR_SIZE;
    int size = 0;
    while (input.read(reinterpret_cast<char *>(&lnum),
sizeof(lnum)))
    {
        input.read(addr_buf, ADDR_SIZE);
        offset_table.push_back(pair<long long int, int>(lnum,
offset * size));
        size += 1;
    }

    sort(offset_table.begin(), offset_table.end(), [](auto a,
auto b)
        { return a.first < b.first; });

    auto start = high_resolution_clock::now();

    // Uniform Binary search
    long long int found = 0;
    int pow = 1;
    int curr_position = 0;
    do
    {
        pow <= 1;
        offset = (size + (pow >> 1)) / pow;

        if (offset_table[curr_position].first == key)
        {
            found = curr_position;
            break;
        }
        else if (offset_table[curr_position].first < key)
        {
            curr_position += offset;
        }
        else
        {
            curr_position -= offset;
        }
    }
}
```

```

    }

    } while (offset > 0);

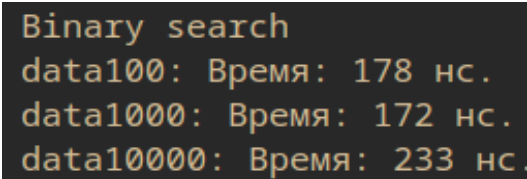
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<nanoseconds>(stop -
start).count();
    cout << file_name << ": Время: " << duration << " нс." <<
endl;

    // Get result
    input.seekg(offset_table[found].second); // Offset bits
based on offset table data
    input.read(reinterpret_cast<char *>(&lnum), sizeof(lnum));
    input.read(addr_buf, ADDR_SIZE);
    string result = to_string(lnum) + " " + string(addr_buf);

    input.close();
    return result;
}

```

4.4 Результаты тестирования



```

Binary search
data100: Время: 178 нс.
data1000: Время: 172 нс.
data10000: Время: 233 нс.

```

Рисунок 4 - Результаты
тестирования бинарного поиска

Таблица 2 — Сравнение тестирования бинарного поиска

Бинарный поиск			
Запуск	1	2	3
100 записей	178 нс	176 нс	166 нс
1000 записей	172 нс	192 нс	251 нс
10000 записей	233 нс	300 нс	308 нс

5 ВЫВОД

Были освоены приёмы работы по применению алгоритмов поиска в таблицах данных. Была разработана программа по поиску записей в бинарном файле с применением различных алгоритмов.

6 ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ

1. Структуры и алгоритмы обработки данных (часть 2): Лекционные материалы / Рысин М. Л. МИРЭА — Российский технологический университет, 2022/23. – 82 с.