МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных технологий

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 9
по дисциплине
«Структуры и алгоритмы обработки данных»

Тема: «Поиск в файле»

Выполнил студент группы ИКБО-18-22 Ракитин В.А.

Принял преподаватель Филатов А.С.

Лабораторная работа выполнена «__»_____202__ г. *(подпись студента)*

«Зачтено» «__»_____202__ г. *(подпись руководителя)*

Москва 2023

1. Цель работы

Получить практический опыт по применению алгоритмов поиска в таблицах данных.

2. Постановка задачи

1. Разработать программу генерации двоичного файла из записей (структура записи определена вариантом). Поле ключа записи в задании варианта подчеркнуто. Файл заполнять данными, используя для датчик псевдослучайных чисел. Ключи записей в файле уникальны.

    a. Для удобства дальнейшей работы в программе выводить количество записанных записей, байт и последнюю запись.

2. Разработать программу поиска записи по ключу в бинарном файле с применением алгоритма линейного поиска.

    a. Провести практическую оценку времени выполнения поиска последней записи в файле объемом 100, 1000, 10 000 и более записей.
    b. Составить таблицу с указанием результатов замера времени.

3. Для оптимизации поиска в файле, разработать программу, создающую в оперативной памяти дополнительную структуру данных, содержащую ключ и ссылку (смещение) на запись в файле.

    a. Разработать функцию, которая принимает на вход ключ, ищет в таблице элемент, соответствующий ключу, и возвращает ссылку на запись в файле. Алгоритм поиска определен в варианте.
    b. Разработать функцию, которая принимает ссылку на запись в файле, считывает ее, применяя механизм прямого доступа к записям файла. И возвращает прочитанную запись как результат.
    c. Провести практическую оценку времени выполнения поиска последней записи в файле объемом 100, 1000, 10 000 и более записей.

d. Составить таблицу с указанием результатов замера времени. Отдельно учесть время, затраченное на создание дополнительной структуры.

4. Провести анализ эффективности разработанного алгоритма поиска по сравнению с линейным.

Вариант №5. Условие задания:

Алгоритм поиска	Интерполяционный поиск
Структура записи файла (ключ – подчеркнутое поле)	Пациент поликлиники: номер карточки – целое число, код хронического заболевания, Фамилия лечащего врача

3. Решение

Бинарный файл – это файл, в котором вся информация записано цифрами «0» и «1». Такой вид файла ещё называют двоичным.

В основе интерполяционного поиска лежит операция интерполирование. Интерполирование – нахождение промежуточных значений величины по имеющемуся дискретному набору известных значений. Интерполяционный поиск работает только с упорядоченными массивами; он похож на бинарный, в том смысле, что на каждом шаге вычисляется некоторая область поиска, которая, по мере выполнения алгоритма, сужается.

Формула, определяющая алгоритм интерполяционного поиска выглядит следующим образом: $mid = left + \frac{(key - A[left]) * (right - left)}{A[right] - A[left]}$

mid – номер элемента, с которым сравнивается значение ключа, key – ключ (искомый элемент), A – массив упорядоченных элементов, left и right – номера крайних элементов области поиска.

Для решения первого упражнения была создана структура Patient, в которой хранятся данные пациента: ключ, код болезни и фамилия врача.

```
struct Patient {
    int numberCard; // ключ
    int chronicDiseaseCode;
    string treatingDoctor;
};
```

Также для решения упражнения была написана функция generateFile, которая создаёт файл. На вход функция получает целое число – количество пациентов. Далее программа записывает данные в файл и выводит размер файла в байтах и данные последнего пациента.

```
int generateFile(int patientNumber) {
    Patient* patient_list = new Patient[patientNumber];
    FILE* file = fopen(fileName, "wb+");
    char Family[][9] = {"Иванов", "Петров", "Сидоров", "Семенов", "Крыжов",
"Кукувов", "Манотов", "Китиров", "Авутилин", "Пронин"};
    int code = rand() % 173 + 1;
    for (int i = 0; i < patientNumber; i++) {
        patient_list[i].numberCard = code;
        patient_list[i].chronicDiseaseCode = rand() % 523 + 1;
        patient_list[i].treatingDoctor = Family[rand() % 10];
        code += 1;
    }
    Patient last = patient_list[patientNumber - 1];
    fwrite(patient_list, sizeof(Patient), patientNumber, file);
    fclose(file);
    cout << endl << "Размер файла: " << sizeof(Patient) * patientNumber << "
байт";
    cout << endl << "Последний пациент: " << last.numberCard << " " <<
last.chronicDiseaseCode
        << " " << last.treatingDoctor;
    return patientNumber;
}
```

Также для решения упражнения была написана функция linearSearch, которая реализует линейный поиск. На вход функция получает два значения: количество пациентов и ключ. Далее функция возвращает данные найденного пациента.

```
Patient linearSearch(int N, int numberCard){
    FILE* file = fopen(fileName, "rb");
    Patient temp;
    for (int i = 0; i < N+1; i++)
    {
        fread(&temp, sizeof(Patient), 1, file);
        if (temp.numberCard == numberCard) {
            return temp;
        }
    }
```

```
    fclose(file);
}
```

Также для решения упражнения была создана структура Table, в которой хранятся ключ и позиция. Она нужна для создания интерполяционного поиска.

```
struct Table {
  int key;
  int offset;
};
```

Также для решения упражнения была написана функция createSearchTable, которая создаёт таблицу. На вход функция получает два значения: структуру и количество пациентов. Далее функция записывает данные в структуру.

```
void createSearchtable(Table*& table, int N) {
    FILE* file = fopen(fileName, "rb");
    table = new Table[N];
    Patient temp;
    for (int i = 0; i < N; i++) {
        fread(&temp, sizeof(Patient), 1, file);
        table[i].key = temp.numberCard;
        table[i].offset = i;
    }
  qsort(table, N, sizeof(Table), comp);
  fclose(file);
}
```

Также для решения упражнения была написана функция interpolationSearch, которая реализует алгоритм интерполяционного поиска. На вход функция получает три значения: структуру, количество пациентов и ключ. Далее функция выводит данные о найденном пациенте.

```
Table interpolationSearch(Table* table, int N, int key){
    int left = 0, right = N - 1;
    while (left <= right && key >= table[left].key && key <=
table[right].key){
        int pos = left + ((key - table[left].key) * (right - left)) /
(table[right].key - table[left].key);
        if (table[pos].key == key) {
            return table[pos];
        }
        if (table[pos].key < key) {
            left = pos + 1;
      }
      else {
            right = pos - 1;
          }
    }
    return table[0];
}
```

При запуске программы первого упражнения, пользователь видит пользовательское меню, где надо выбрать, какую задачу надо решить.

```
1 - Добавить информацию в файл
2 - Поиск записи по ключу линейным поиском
3 - Поиск записи по ключу интерполционным поиском
0 - Завершить работу программы
Выберите 1, 2, 3, 0:
```

Рисунок 1. Интерфейс программы

4. Тестирование

Протестируем программой решение первого упражнения. Введём следующие количества данных в файле: 100, 1000, 10000, 25000. Составим таблицу с результатами тестирования линейного поиска.

```
1 - Добавить информацию в файл
2 - Поиск записи по ключу линейным поиском
3 - Поиск записи по ключу интерполционным поиском
0 - Завершить работу программы
Выберите 1, 2, 3, 0: 1

Сколько пациентов добавить? 100

Размер файла: 4800 байт
Последний пациент: 140 342 Семенов
1 - Добавить информацию в файл
2 - Поиск записи по ключу линейным поиском
3 - Поиск записи по ключу интерполционным поиском
0 - Завершить работу программы
Выберите 1, 2, 3, 0: 2
Введите номер ключа: 140

Найден пациент: 140 342 Семенов
Время поиска: 0.0001759
```

Рисунок 2. Решение программой упражнения

```
1 - Добавить информацию в файл
2 - Поиск записи по ключу линейным поиском
3 - Поиск записи по ключу интерполционным поиском
0 - Завершить работу программы
Выберите 1, 2, 3, 0: 1

Сколько пациентов добавить? 1000

Размер файла: 48000 байт
Последний пациент: 1040 371 Пронин
1 - Добавить информацию в файл
2 - Поиск записи по ключу линейным поиском
3 - Поиск записи по ключу интерполционным поиском
0 - Завершить работу программы
Выберите 1, 2, 3, 0: 2
Введите номер ключа: 1040

Найден пациент: 1040 371 Пронин
Время поиска: 0.0002937
```

Рисунок 3. Решение программой упражнения

```
1 - Добавить информацию в файл
2 - Поиск записи по ключу линейным поиском
3 - Поиск записи по ключу интерполционным поиском
0 - Завершить работу программы
Выберите 1, 2, 3, 0: 1

Сколько пациентов добавить? 10000

Размер файла: 480000 байт
Последний пациент: 10040 302 Семенов
1 - Добавить информацию в файл
2 - Поиск записи по ключу линейным поиском
3 - Поиск записи по ключу интерполционным поиском
0 - Завершить работу программы
Выберите 1, 2, 3, 0: 2
Введите номер ключа: 10040

Найден пациент: 10040 302 Семенов
Время поиска: 0.0013736
```

Рисунок 4. Решение программой упражнения

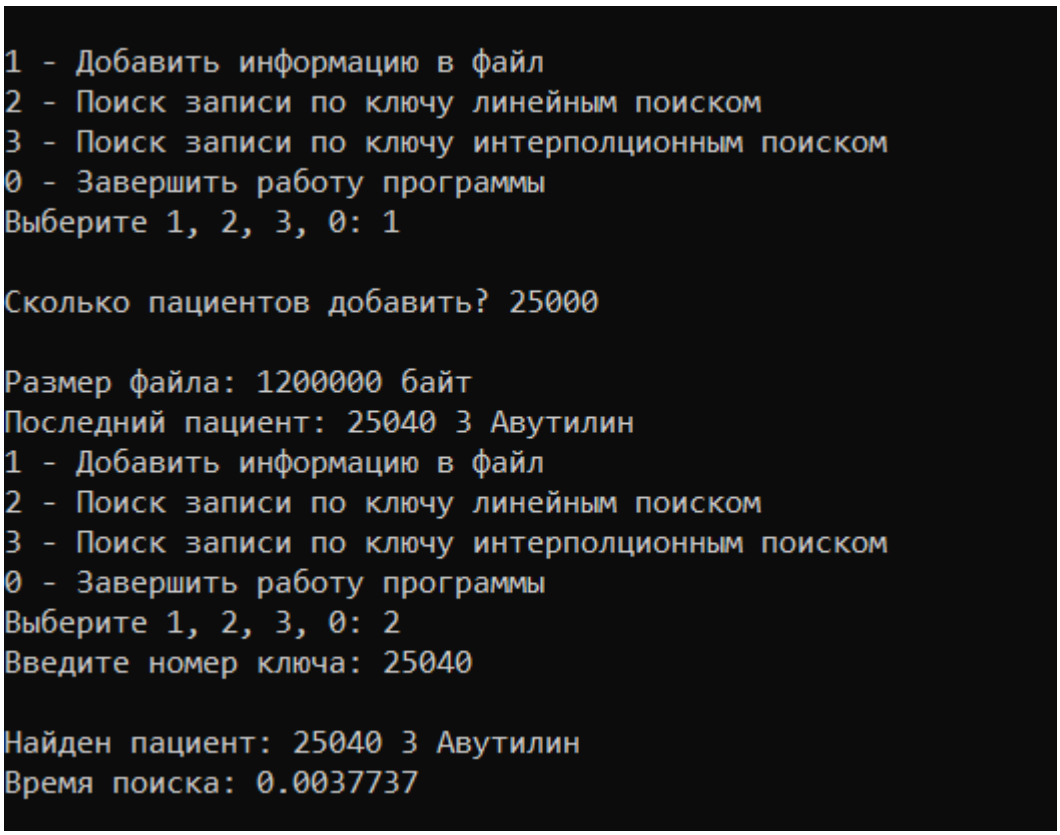

Рисунок 5. Решение программой упражнения

Таблица 1. Тестирование линейного поиска

N	**t, c.**
100	0.00018
1000	0.00029
10000	0.00138
25000	0.00377

Аналогично составим таблицу с результатами интерполяционного поиска, используя те же самые файлы.

```
Размер файла: 4800 байт
Последний пациент: 140 342 Семенов
1 - Добавить информацию в файл
2 - Поиск записи по ключу линейным поиском
3 - Поиск записи по ключу интерполционным поиском
0 - Завершить работу программы
Выберите 1, 2, 3, 0: 3
Введите номер ключа: 140

Найден пациент: 140 342 Семенов
Время создания таблицы: 0.0001789
Время поиска: 0.000116
```

Рисунок 6. Решение программой упражнения

```
Размер файла: 48000 байт
Последний пациент: 1040 371 Пронин
1 - Добавить информацию в файл
2 - Поиск записи по ключу линейным поиском
3 - Поиск записи по ключу интерполционным поиском
0 - Завершить работу программы
Выберите 1, 2, 3, 0: 3
Введите номер ключа: 1040

Найден пациент: 1040 371 Пронин
Время создания таблицы: 0.0005289
Время поиска: 0.0001038
```

Рисунок 7. Решение программой упражнения

```
Размер файла: 480000 байт
Последний пациент: 10040 302 Семенов
1 - Добавить информацию в файл
2 - Поиск записи по ключу линейным поиском
3 - Поиск записи по ключу интерполционным поиском
0 - Завершить работу программы
Выберите 1, 2, 3, 0: 3
Введите номер ключа: 10040

Найден пациент: 10040 302 Семенов
Время создания таблицы: 0.0046248
Время поиска: 0.0001138
```

Рисунок 8. Решение программой упражнения

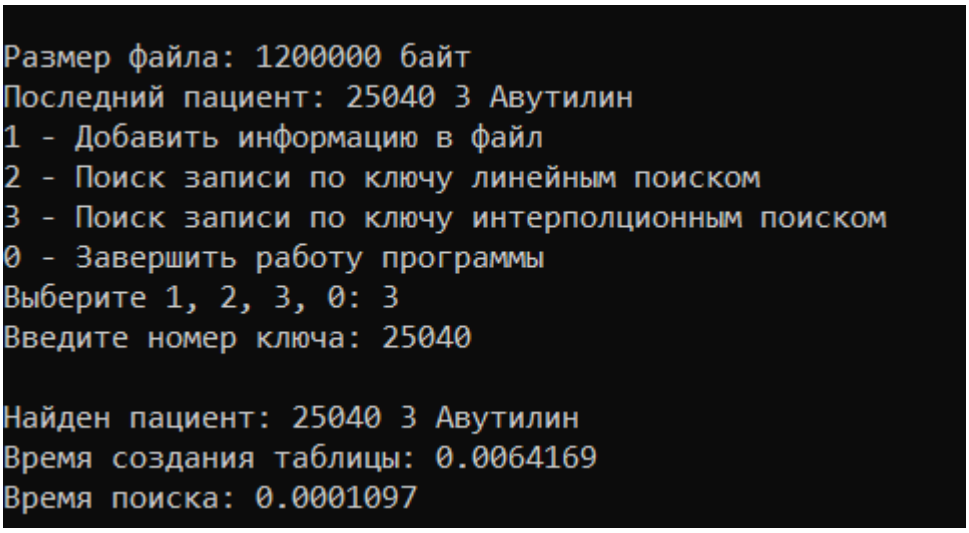

Рисунок 9. Решение программой упражнения

Таблица 2. Тестирование интерполяционного поиска

N	**t создания таблицы, с.**	**t поиска, с.**
100	0.000179	0.000116
1000	0.000529	0.000104
10000	0.004625	0.000114
25000	0.006417	0.000110

Из результатов выполнения программы видно, что интерполяционный поиск быстрее чем линейный, но это преимущество в скорости нивелируется при создании доп. структуры и ее сортировки. То есть, если записи в файле уже отсортированы, то интерполяционный поиск намного лучше, чем линейный.

5. Вывод

В результате работы я:

1. Получил навыки создания бинарного файла
2. Освоил алгоритмы поиска текста в бинарном файле

6. Исходный код программы

```
#include <iostream>
```

```
#include <algorithm>
#include <windows.h>
#include <chrono>
#include <fstream>
#include <random>
#pragma warning(disable : 4996)
using namespace std;
const char* fileName = "patientFile.bin";

struct Patient {
    int numberCard; // ключ
    int chronicDiseaseCode;
    string treatingDoctor;
};

// Структура элемента таблицы, содержащей ключ и ссылку на запись в файле
struct Table {
  int key;
  int offset;
};

int generateFile(int patientNumber) {
    Patient* patient_list = new Patient[patientNumber];
    FILE* file = fopen(fileName, "wb+");
    char Family[][9] = {"Иванов", "Петров", "Сидоров", "Семенов", "Крыжов",
"Кукувов", "Манотов", "Китиров", "Авутилин", "Пронин"};
    int code = rand() % 173 + 1;
    for (int i = 0; i < patientNumber; i++) {
        patient_list[i].numberCard = code;
        patient_list[i].chronicDiseaseCode = rand() % 523 + 1;
        patient_list[i].treatingDoctor = Family[rand() % 10];
        code += 1;
    }
    Patient last = patient_list[patientNumber - 1];
    fwrite(patient_list, sizeof(Patient), patientNumber, file);
    fclose(file);
    cout << endl << "Размер файла: " << sizeof(Patient) * patientNumber << "
байт";
    cout << endl << "Последний пациент: " << last.numberCard << " " <<
last.chronicDiseaseCode
        << " " << last.treatingDoctor;
    return patientNumber;
}

Patient linearSearch(int N, int numberCard){
    FILE* file = fopen(fileName, "rb");
    Patient temp;
    for (int i = 0; i < N+1; i++)
    {
        fread(&temp, sizeof(Patient), 1, file);
        if (temp.numberCard == numberCard) {
            return temp;
        }
    }
    fclose(file);
}

int comp(const void* a, const void* b) {
    return (*(int*)a - *(int*)b);
}

void createSearchtable(Table*& table, int N) {
    FILE* file = fopen(fileName, "rb");
    table = new Table[N];
```

```
    Patient temp;
    for (int i = 0; i < N; i++) {
        fread(&temp, sizeof(Patient), 1, file);
        table[i].key = temp.numberCard;
        table[i].offset = i;
    }
  qsort(table, N, sizeof(Table), comp);
  fclose(file);
}

Table interpolationSearch(Table* table, int N, int key){
    int left = 0, right = N - 1;
    while (left <= right && key >= table[left].key && key <=
table[right].key){
        int pos = left + ((key - table[left].key) * (right - left)) /
(table[right].key - table[left].key);
        if (table[pos].key == key) {
            return table[pos];
        }
        if (table[pos].key < key) {
            left = pos + 1;
      }
      else {
            right = pos - 1;
        }
    }
    return table[0];
}


Patient getPatient(int offset){
    FILE* file = fopen(fileName, "rb");
    fseek(file, (offset) * sizeof(Patient), SEEK_SET);
    Patient temp;
    fread(&temp, sizeof(Patient), 1, file);
    fclose(file);
    return temp;
}

int main() {
    setlocale(LC_ALL, "Rus");
    int N;
    while (true) {
        int choise;
        cout << endl << "1 - Добавить информацию в файл" << endl
            << "2 - Поиск записи по ключу линейным поиском" << endl
            << "3 - Поиск записи по ключу интерполционным поиском" << endl
            << "0 - Завершить работу программы" << endl
            << "Выберите 1, 2, 3, 0: "; cin >> choise;
        switch (choise)
        {
        case(1):
        {
            int patientNumber;
            cout << endl << "Сколько пациентов добавить? ";
            cin >> patientNumber;
            N = generateFile(patientNumber);
            break;
        }
        case(2):
        {
            int key1;
            cout << "Введите номер ключа: "; cin >> key1;
            auto start = std::chrono::steady_clock::now();
```

```
            Patient foundPatient = linearSearch(N, key1);
            auto end = std::chrono::steady_clock::now();
            std::chrono::duration<double> elapsed_seconds = end - start;
            cout << endl << "Найден пациент: " << foundPatient.numberCard <<
" " << foundPatient.chronicDiseaseCode
                << " " << foundPatient.treatingDoctor << endl << "Время
поиска: " << elapsed_seconds.count() << endl;
            break;
        }
        case(3):
        {
            int key2;
            Table* searchtable{};
            cout << "Введите номер ключа: "; cin >> key2;
            auto start = std::chrono::steady_clock::now();
            createSearchtable(searchtable, N+1);
            auto end = std::chrono::steady_clock::now();
            std::chrono::duration<double> elapsed_seconds = end - start;
            auto start1 = std::chrono::steady_clock::now();
            Table Found = interpolationSearch(searchtable, N, key2);
            Patient foundPatient = getPatient(Found.offset);
            auto end1 = std::chrono::steady_clock::now();
            std::chrono::duration<double> elapsed_seconds1 = end1 - start1;
            cout << endl << "Найден пациент: " << foundPatient.numberCard <<
" " << foundPatient.chronicDiseaseCode<< " " << foundPatient.treatingDoctor;
            cout << endl << "Время создания таблицы: " <<
elapsed_seconds.count();
            cout << endl << "Время поиска: " << elapsed_seconds1.count() <<
endl;
            break;
        }
        case(0):
            return 0;
            break;
        default:
            cout << endl << "Вы ввели некорректное значение!" << endl;
            break;
        }
    }
}
```