|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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 № 5.2** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Алгоритмы поиска в таблице при работе с данными из файла»** | |
|  | |
| Выполнил студент группы ИКБО-10-23 | Харитонов А.Н. |
| Принял преподаватель | Макеева О.В. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2024

# **Цель работы**

Получить практический опыт по применению алгоритмов поиска в таблицах данных.

# **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.

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

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

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

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

* 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.
* Разработать функцию, которая принимает ссылку на запись в файле, считывает ее, применяя механизм прямого доступа к записям файла. И возвращает прочитанную запись как результат.
* Провести практическую оценку времени выполнения поиска последней записи в файле объемом 100, 1000, 10 000 и более записей.
* Составить таблицу с указанием результатов замера времени. Отдельно учесть время, затраченное на создание дополнительной структуры.

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

Вариант №13. Условие задания:

|  |  |
| --- | --- |
| Алгоритм поиска | Бинарный однородный с использованием таблицы смещений |
| Структура записи файла | Студент: номер зачетной книжки, номер группы, ФИО |

# **Решение**

Бинарный поиск – классический алгоритм поиска элемента в отсортированном массиве, использующий метод дробления массива на половины.

Бинарный однородный поиск – модификация обычного бинарного поиска. Он принимает два параметра – текущая позиция и величина изменения. Работает по схожему принципу с бинарным, изначальное положение устанавливается на середину массива, а шаг – (n / 2), где n – длина массива. С каждым шагом величина изменения уменьшается вдвое, а положение движется вправо или влево на (величина изменения / 2) позиций. Данный поиск продолжается до тех пор, пока величина изменений не равна 0 (возврат -1) или же пока не будет найден искомый элемент.

Для данной работы были написаны 2 структуры:

Struct card, хранящая в себе StudNumber, группу и ФИО

*struct* card {

*int* StudNumber;

*char* group[20];

*char* name[50];

    card(*int* *StudNumber*, const *char*\* *group\_in*, const *char*\* *name\_in*)

        : StudNumber(*StudNumber*) {

        strncpy\_s(group, *group\_in*, \_TRUNCATE);

        strncpy\_s(name, *name\_in*, \_TRUNCATE);

    }

    card() = default;

    string toString() {

        return format("StudNumber = {:8}\tGroup: {:10}\tFIO:  {:50}", StudNumber, group, name);

    }

};

Struct cardSearch, хранящая в себе ключ и позицию card в списке

*struct* cardSearch {

*int* key;

*int* pos;

};

Для решения поставленной задачи необходимо создать изначальный массив, для этого была написана функция generateData, принимающая на вход массив, который надо заполнить и количество студентов.

card generateData(card\* *cardArray*, *int* *cardsCounter*)

{

    string groups[5] = { "IKBO-10-23", "IKBO-11-23", "IKBO-12-23", "IKBO-13-23", "IKBO-14-23" };

    string names[5] = { "Kharitonov Arseniy", "Antonov Nikolai", "Moiseev Artur", "Gribov Ivan", "Radoms Anton" };

    card tempCard;

    vector<*int*> randoms = generateRandomNumbers(*cardsCounter*, 100'000'000, 900'000'000);

    for (*int* i = 0; i < *cardsCounter*; i++) {

        tempCard = card(randoms[i], groups[generateRandomNumbers(1, 100'000'000, 900'000'000)[0] % 5].c\_str(), names[generateRandomNumbers(1, 100'000'000, 900'000'000)[0] % 5].c\_str());

*cardArray*[i] = tempCard;

    }

    return tempCard;

}

Для реализации поиска записи по ключу в файле путем линейного поиска была написана функция lineraSearch, принимающая на вход объект класса fstream, кол-во студентов и необходимый номер. Данная функция возвращает объект класса card, если необходимой записи не найдено – возвращает запись StudNumber равным -1.

card linearSearch(fstream& *fs*, *int* *cardCounter*, *int* *StudNumber*)

{

*fs*.seekg(0);

    card tempCard;

    for (*int* i = 0; i < *cardCounter*; i++) {

*fs*.read((*char*\*)&tempCard, sizeof(card));

        if (tempCard.StudNumber == *StudNumber*) return tempCard;

    }

    return card(-1, "NONE", "NONE");

}

Для оптимизации поиска в файле были написаны функции createTable, создающая отсортированную таблицу из структур cardSearch, tableSort, сортирующая таблицу, modifiedBinarySearch, осуществляющую бинарный поиск.

Функция tableSort принимает на вход объект класса cardSearch и кол-во студентов, сортирующая таблицу методом простого выбора.

*void* tableSort(cardSearch\* *table*, *int* *cardCounter*)

{

    for (*int* i = 0; i < *cardCounter* - 1; i++)

    {

*int* ind = i;

        for (*int* j = i + 1; j < *cardCounter*; j++)

        {

            if (*table*[j].key < *table*[ind].key) ind = j;

        }

        cardSearch tempCard = *table*[i];

*table*[i] = *table*[ind];

*table*[ind] = tempCard;

    }

}

Функция createTable принимает на вход объект класса fstream и количество студентов. В данной функции обеспечивается считывание файла и создание структуры класса cardSearch, а также список класса card, для дальнейшего обращения к найденной записи.

cardSearch\* createTable(fstream& *fs*, *int* *cardCounter*, card\* *cardArray*)

{

*fs*.seekg(0);

    cardSearch\* table = new cardSearch[*cardCounter*];

    card tempCard;

    for (*int* i = 0; i < *cardCounter*; i++) {

*fs*.read((*char*\*)&tempCard, sizeof(card));

*cardArray*[i] = tempCard;

        table[i].key = tempCard.StudNumber;

        table[i].pos = i;

    }

    tableSort(table, *cardCounter*);

    return table;

}

Функция modifiedBinarySeach принимает на вход таблицу класса cardSearch, искомый ключ, позицию, шаг и количество студентов.

cardSearch modifiedBinarySearch(cardSearch\* *table*, *int* *needKey*, *int* *position*, *int* *delta*, *int* *cardCounter*)

{

    if (*delta* == 0) return cardSearch(-1, -1);

    if (*table*[*position*].key == *needKey*) return *table*[*position*];

    if (*position* != 0) if (*table*[*position* - 1].key == *needKey*) return *table*[*position* - 1];

    if (*position* != *cardCounter* - 1) if (*table*[*position* + 1].key == *needKey*) return *table*[*position* + 1];

*delta* /= 2;

    if (*table*[*position*].key > *needKey*)

        return modifiedBinarySearch(*table*, *needKey*, *position* - *delta*, *delta*, *cardCounter*);

    else

        return modifiedBinarySearch(*table*, *needKey*, *position* + *delta*, *delta*, *cardCounter*);

}

При запуске программы программа сразу же начинает заполнять массив, а также выводит последнюю запись в массиве.

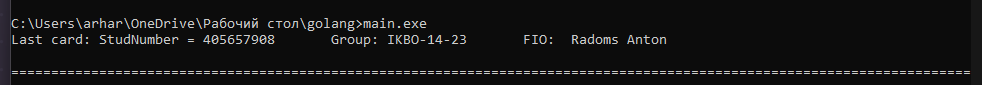


Рисунок 1. Интерфейс программы

# **Тестирование**

Протестируем программу, для этого запустим ее и введем номер последней записи, немного подождав мы увидим результат поиска, а также время, за которое он произошел. Протестируем поиск на 1000, 10000 и 100000 записях.

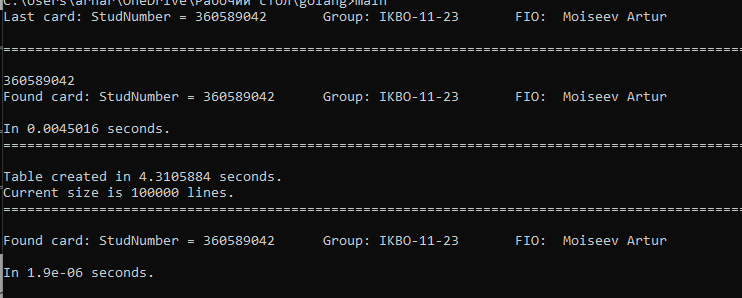


Рисунок 2. Тестирование работы программы при 100000 записях

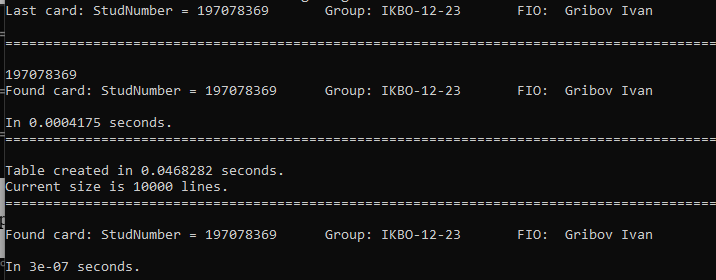


Рисунок 3. Тестирование работы программы при 10000 записях

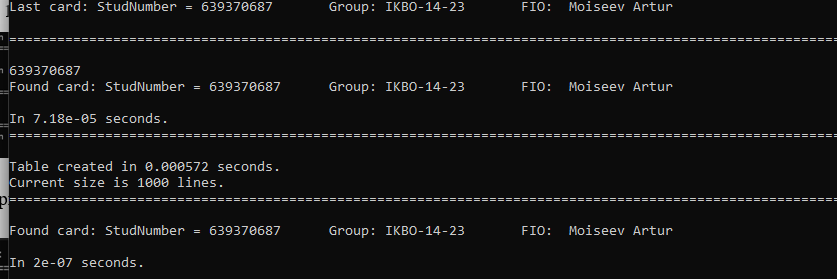


Рисунок 4. Тестирование работы программы при 1000 записях

Таблица 1. Табличное представление времени поиска

|  |  |  |
| --- | --- | --- |
| Кол-во записей | Линейный поиск(мс) | Улучшенный бинарный поиск(мс) |
| 1000 | 0,072 | 0,0002 |
| 10000 | 0,4 | 0,0003 |
| 100000 | 4,5 | 0,0019 |

Из результатов выполнения программы видно:

* Написанная структура в связке с улучшенным алгоритмом поиска дает огромный прирост в времени работы нашей системы.

# **Вывод**

В результате выполнения работы я получил практический опыт по применению алгоритмов поиска в таблицах данных.