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 № 3

по дисциплине

«Структуры и алгоритмы обработки данных»

Тема: «Разработка и программная реализация задач поиска данных в таблицах с применением механизма хеширования.»

Выполнил студент группы ИКБО-11-22 Гришин А. В.

Принял преподаватель Скворцова Л. А.

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

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

Москва 2023

Содержание

1. Цель работы

Получить навыки по разработке хеш-таблиц и их применении при поиске данных в других структурах данных (файлах).

2. Задание

2.1 Постановка задачи

Разработать приложение, которое использует хеш-таблицу для организации прямого доступа к записям двоичного файла, реализованного в практической работе 2.

Метод разрешения коллизии представлен в вашем варианте задания в таб.1.

Требования к выполнению

1. Создать приложение и включить в него три заголовочных файла: управление хеш-таблицей, управление двоичным файлом (практическая работа 2), управление двоичным файлом посредством хеш-таблицы. Имена заголовочным файлам определите сами. Подключите заголовочные файлы к приложению.

2. Для обеспечения прямого доступа к записи в файле элемент хеш-таблицы должен включать обязательные поля: ключ записи в файле, номер записи с этим ключом в файле. Элемент может содержать другие поля, требующиеся методу (указанному в вашем варианте), разрешающему коллизию.

3. Управление хеш-таблицей.

1) Определить структуру элемента хеш-таблицы и структуру хеш-таблицы в соответствии с методом разрешения коллизии, указанном в варианте. Определения разместить в соответствующем заголовочном файле.

Все операции управления хеш-таблицей размещать в этом заголовочном файле.

2) Тестирование операций выполнять в функции main приложения по мере их реализации. После тестирования всех операций, создать в заголовочном файле функцию с именем testHeshT переместить в нее содержание функции main, проверить, что приложение выполняется. Разработать операции по управлению хеш-таблицей.

3) Разработать хеш-функцию (метод определить самостоятельно), выполнить ее тестирование, убедиться, что хеш (индекс элемента таблицы) формируется верно.

4) Разработать операции: вставить ключ в таблицу, удалить ключ из таблицы, найти ключ в таблице, рехешировать таблицу. Каждую операцию тестируйте по мере ее реализации.

5) Подготовить тесты (последовательность значений ключей), обеспечивающие:

− вставку ключа без коллизии

− вставку ключа и разрешение коллизии

− вставку ключа с последующим рехешированием

− удаление ключа из таблицы

− поиск ключа в таблице

Примечание. Для метода с открытым адресом подготовить тест для поиска ключа, который размещен в таблице после удаленного ключа, с одним значением хеша для этих ключей.

6) Выполнить тестирование операций управления хеш-таблицей. При тестировании операции вставки ключа в таблицу предусмотрите вывод списка индексов, которые формируются при вставке элементов в таблицу.

4. Управление двоичным файлом.

Операции управления двоичным файлом: создание двоичного файла из текстового, добавить запись в двоичный файл, удалить запись с заданным ключом из файла, прочитать запись файла по заданному номеру записи.

Примечание. Эти операции должны быть отлажены в практической работе 2, или уже в этой работе, если их пока нету.

Структура записи двоичного файла и все операции, по управлению файлом, должны быть размещены в соответствующем заголовочном файле.

Выполнить тестирование операций в main приложения, и содержание функции main переместить в соответствующую функцию заголовочного файла с именем testBinF.

5. Управление файлом посредством хеш-таблицы.

В заголовочный файл управления файлом посредством хеш-таблицы подключить заголовочные файлы: управления хеш-таблицей, управления двоичным файлом. Реализовать поочередно все перечисленные ниже операции в этом заголовочном файле, выполняя их тестирование из функции main приложения. После разработки всех операций выполнить их комплексное тестирование.

Разработать и реализовать операции.

1) Прочитать запись из файла и вставить элемент в таблицу (элемент включает: ключ и номер записи с этим ключом в файле, и для метода с открытой адресацией возможны дополнительные поля).

2) Удалить запись из таблицы при заданном значении ключа и соответственно из файла.

3) Найти запись в файле по значению ключа (найти ключ в хеш-таблице, получить номер записи с этим ключом в файле,выполнить прямой доступ к записи по ее номеру).

4) Подготовить тесты для тестирования приложения: Заполните файл небольшим количеством записей.

− Включите в файл записи как не приводящие к коллизиям, так и приводящие.

− Обеспечьте включение в файл такого количества записей, чтобы потребовалось рехеширование.

Заполните файл большим количеством записей (до 1 000 000).

− Определите время чтения записи с заданным ключом: для первой записи файла, для последней и где-то в середине. Убедитесь (или нет), что время доступа для всех записей одинаково.

2.2 Код программы (персональный вариант 15)

Помимо функций, которые были реализованы в практической работе №2 (такие как добавление записи, удаление записи и т.д.) были добавлены следующие функции.

Структура записи двоичного файла (вариант 15): дата рождения, имя (рис. 1).

```
struct FriendRecord {
    char dateOfBirth[11];
    char name[50];
};
```

Рисунок 1 – структура FriendRecord

Функция insertDataToHashTable записывает данные бинарного файла в хэш-таблицу (рис. 2).

```
void insertDataToHashTable(string binaryFilename) {
    ifstream binaryFile(binaryFilename, ios::binary);
    if (!binaryFile.is_open()) {
        cout << "Error opening file " << binaryFilename << endl;
        return;
    }

    FriendRecord record;

    while (binaryFile.read(reinterpret_cast<char*>(&record), sizeof(FriendRecord))) {
        string result = (record.dateOfBirth);
        hashTable.insert(result, pos);

        pos++;
    }

    cout << endl;
    binaryFile.close();
}
```

Рисунок 2 – Функция insertDataToHashTable

Функция HashTable::hash используется для хэширования ключа (рис. 3).

```
int HashTable::hash(const string& key) {
    unsigned int sum = 0;
    for (char c : key) {
        sum += static_cast<int>(c);
    }
    return sum % TABLE_SIZE;
}
```

Рисунок 3 – Функция HashTable::hash

Функция HashTable::insert реализует тип хэширования «Открытый адрес (смещение на 1)» (рис. 4).

Тип хэширования "Открытый адрес (смещение на 1)" представляет собой метод разрешения коллизий в хэш-таблице, где элементы, имеющие одинаковое значение хэша, размещаются в той же таблице, смещаясь на фиксированное смещение (обычно 1) от исходной позиции, до тех пор, пока не будет найдена свободная ячейка.

```
void HashTable::insert(const string& key, int pos) {
    int index = hash(key);
    int originalIndex = index;

    while (table[index].isOccupied) {
        index = (index + 1) % TABLE_SIZE;
        if (index == originalIndex) {
            return;
        }
    }

    table[index].key = key;
    table[index].pos = pos;
    table[index].isOccupied = true;
}
```

Рисунок 4 – Функция HashTable::insert

Функция HashTable::find используется для поиска элемента по ключу в хэш-таблице (рис. 5)

```
int HashTable::find(const string& key) {
    int index = hash(key);
    int originalIndex = index;

    while (table[index].isOccupied) {
        if (table[index].key == key) {
            return table[index].pos;
        }
        index = (index + 1) % TABLE_SIZE;
        if (index == originalIndex) {
            break;
        }
    }
    return -1;
}
```

Рисунок 5 – Функция HashTable::find

Функция HashTable::remove используется для удаления элемента из хэш-таблицы по ключу (рис. 6)

```
void HashTable::remove(const string& key) {
    int index = find(key);
    if (index != -1) {
        table[index].isOccupied = false;
    }
}
```

Рисунок 6 – Функция HashTable::remove

Функция findLineByPos реализует поиск записи в бинарном файле с именем “binaryFilename” по заданной позиции “pos” (рис. 7). Поиск

реализуется посредством перемещения указателя в файле, чтения соответствующей записи и возврата информации о записи в виде строки.

```
string findLineByPos(string binaryFilename, int pos) {
    ifstream binaryFile(binaryFilename, ios::binary);

    binaryFile.seekg(pos * sizeof(FriendRecord), ios::beg);
    FriendRecord record;
    binaryFile.read(reinterpret_cast<char*>(&record), sizeof(FriendRecord));
    binaryFile.close();
    return string() + record.dateOfBirth + " " + record.name;
}
```

Рисунок 7 – Функция findLineByPos

Функция addNewLine реализует добавление новой записи в бинарный файл и хэш-таблицу (рис. 8).

```
void addNewLine(string newLine, string binaryFilename) {
    ofstream binaryFile(binaryFilename, ios::binary | ios::app);
    if (!binaryFile.is_open()) {
        cout << "Error opening file " << binaryFilename << endl;
        return;
    }

    FriendRecord record;
    istringstream iss(newLine);
    iss >> record.dateOfBirth >> record.name;

    binaryFile.write(reinterpret_cast<char*>(&record), sizeof(FriendRecord));
    string result = record.dateOfBirth;
    hashTable.insert(result, pos);
    pos++;
    binaryFile.close();
}
```

Рисунок 8 – Функция addNewLine

2.3. Тестирование программы.

```
Enter filename: friends.txt
File friends.txt opened successfully
1. Converting text file to binary file
2. Converting binary file to text file
3. Print data from binary file
4. Insert data to hash table
5. Find data in hash table by key
6. Add new line to binary file and hash table
7. Remove line from binary file and hash table
0. Exit
Enter command: 1
Text file converted to binary file
1. Converting text file to binary file
2. Converting binary file to text file
3. Print data from binary file
4. Insert data to hash table
5. Find data in hash table by key
6. Add new line to binary file and hash table
7. Remove line from binary file and hash table
0. Exit
Enter command: 2
Binary file converted to text file
1. Converting text file to binary file
2. Converting binary file to text file
3. Print data from binary file
4. Insert data to hash table
5. Find data in hash table by key
6. Add new line to binary file and hash table
7. Remove line from binary file and hash table
0. Exit
Enter command: 3
01.01.1990 Daniel
15.03.1985 Alice
22.11.2022 Ilya
10.12.1988 Emily
05.06.1992 Michael
11.11.2011 Ignat
11.12.1999 Chloe
```

Рисунок 9 — вывод программы.

```
1. Converting text file to binary file
2. Converting binary file to text file
3. Print data from binary file
4. Insert data to hash table
5. Find data in hash table by key
6. Add new line to binary file and hash table
7. Remove line from binary file and hash table
0. Exit
Enter command: 4

Data inserted to hash table
1. Converting text file to binary file
2. Converting binary file to text file
3. Print data from binary file
4. Insert data to hash table
5. Find data in hash table by key
6. Add new line to binary file and hash table
7. Remove line from binary file and hash table
0. Exit
Enter command: 5
Enter key: 22.11.2022
22.11.2022 Ilya
Time: 0.00002020 seconds

1. Converting text file to binary file
2. Converting binary file to text file
3. Print data from binary file
4. Insert data to hash table
5. Find data in hash table by key
6. Add new line to binary file and hash table
7. Remove line from binary file and hash table
0. Exit
Enter command: 6
Enter new line: 01.01.1970 Ivan
Your line: 01.01.1970 Ivan
```

Рисунок 10 — вывод программы.

```
1. Converting text file to binary file
2. Converting binary file to text file
3. Print data from binary file
4. Insert data to hash table
5. Find data in hash table by key
6. Add new line to binary file and hash table
7. Remove line from binary file and hash table
0. Exit
Enter command: 7
Enter key: 10.12.1988
Line removed
```

Рисунок 11 — вывод программы.

2.4. Входные данные программы.

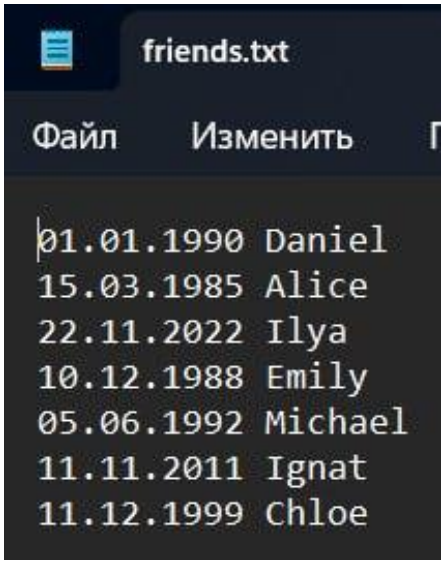

Рисунок 12 — входные данные программы.