МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ 6_1
по дисциплине
«Структуры и алгоритмы обработки данных»

Тема. Применение хеш-таблицы для оптимизации поиска данных.

Выполнил студент группы ИКБО-60-23 Шеенко В.А

Принял старший преподаватель Скворцова Л.А.

Москва 2024

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Получить навыки по разработке программы управления хеш-таблицей и её адаптации для поиска данных в других структурах данных (файлах).

Требования по выполнению практической работы.

Данная практическая работа включает два задания.

Первое задание направлено на моделирование процесса поиска по ключу посредством хеш-таблицы, при этом данные, в которых будет выполняться поиск, будут представлены массивом записей, содержащих ключ и связанные с ним данные. В задании требуется выполнить разработку кода создания и управления хеш-таблицей в отдельном файле заголовка.

Второе задание заключается в применении хеш-таблицы для оптимизации алгоритма поиска в бинарном файле с записями фиксированного размера (который тоже можно считать массивом во внешней памяти). При этом код первого задания адаптируется в соответствии с новой структурой, в которой будет осуществляться поиск.

Индивидуальный вариант №28:

Таблица 1 – Задание индивидуального варианта

Открытый адрес (смещение на 1)	1) Учет нарушений ПДД. Структура записи о нарушении ПДД: номер автомобиля, фамилия и инициалы владельца, модель, дата нарушения, место нарушения (текстом), статья (КоАП), наказание (сумма штрафа).

2. ЗАДАНИЕ №1

2.1 Требования к выполнению задания

1. Разработать программу и определить в ней тестовый (исходный) массив из с 20 элементов (записей) (структура элемента определена вариантом). Инициализировать массив значениями. Этот массив будут представлять модель структуры данных, в которой будет выполняться поиск с использованием хеш-таблицы.

2. Разработать и реализовать в созданной программе операции управления хеш-таблицей.

2.1. Структура элемента хеш-таблицы. Для обеспечения прямого доступа к элементу массива элемент хеш-таблицы должен включать обязательные поля: − ключ записи; − индекс элемента массива, содержащий запись с этим ключом; − признак включения ключа при коллизии (назовем его prK). Элемент может содержать другие поля, требующиеся методу (указанному в вашем варианте), разрешающему коллизию.

2.2. Создать файл заголовка для определения требующихся имен переменных, констант и операций управления хеш-таблицей. Операции должны быть реализованы в формате функций.

2.2.1. Определить в созданном файле заголовка:

1) структуру элемента хеш-таблицы;

2) структуру хеш-таблицы для 11 элементов в соответствии с методом разрешения коллизии, указанном в варианте и саму таблицу;

3) прототипы операций по управлению хеш-таблицей: − хеш-функция (алгоритм определить самостоятельно), выполнить ее тестирование, убедиться, что хеш (индекс элемента таблицы) формируется верно; − вставить ключ в таблицу (учесть заполнение поля prK); − удалить ключ из таблицы; − найти ключ в таблице и вернуть ее номер (смещение) в файле; − вывод таблицы; −

рехешировать таблицу; − создать хеш-таблицу для данных исходного массива.

2.2.2. Выполнить реализацию всех операций, коды сохранить в этом же файле заголовка.

2.3. Для отладки операций подготовить в тестовом массиве значения, которые обеспечат: − вставку ключа без коллизии; − вставку ключа и разрешение коллизии; − вставку ключа с последующим рехешированием; − удаление ключа из таблицы; − поиск ключа в таблице и возвращение всех данных записи.

2.4. Тестирование выполнения операций выполнять в функции main программы по мере их реализации. После тестирования всех операций, создать в заголовочном файле функцию с именем testHeshT переместить в нее содержание функции main. Код в main заменить на вызов функции testHeshT и проверить, что программа корректно выполняется.

2.5. Выполнить тестирование операций управления хеш-таблицей. По завершении операций вставки и удаления должна выводиться таблица.

2.2 Тестовый пример

На рис. 1 представлен динамический двумерный массив, представляющий тестовые данные.

```
{"A123BC77", "Ivanov I.I.", "Toyota Camry", "15.09.2024", "ul. Lenina, d. 12", "12.9 ch.2", 500},
{"B456OR77", "Petrov P.P.", "Hyundai Solaris", "18.09.2024", "pr. Mira, d. 8", "12.16 ch.3", 1000},
{"C789KN77", "Smirnov A.A.", "Kia Rio", "20.09.2024", "ul. Pushkina, d. 5", "12.5 ch.1", 1500},
{"D321OR99", "Sidorov V.V.", "Ford Focus", "12.09.2024", "ul. Gagarina, d. 7", "12.8 ch.2", 2000},
{"E654TR77", "Kuznetsov N.N.", "Nissan Qashqai", "14.09.2024", "pr. Lomonosova, d. 22", "12.15 ch.4", 700},
{"F987KR77", "Fedorov A.S.", "Mazda 3", "16.09.2024", "ul. Chaykovskogo, d. 9", "12.19 ch.1", 1200},
{"G432NC77", "Morozov D.M.", "Volkswagen Polo", "19.09.2024", "pr. Sakharova, d. 4", "12.16 ch.1", 500},
{"H654TM77", "Vasiliev I.V", "Skoda Octavia", "13.09.2024", "ul. Tolstogo, d. 10", "12.9 ch.2", 2500},
{"K987RP77", "Nikolaev S.S", "Renault Duster", "17.09.2024", "ul. Chekhova, d. 15", "12.5 ch.2", 1800},
{"L123BC77", "Zaitsev M.M.", "Lada Vesta", "21.09.2024", "ul. Gorkogo, d. 11", "12.6 ch.1", 900},
{"N038EK34", "Korolev V.V.", "Mazda 6", "10.05.2024", "ul. Tolstogo, d. 15", "12.6 p.1", 2312},
{"P743YA23", "Volkov S.A.", "Ford Explorer", "27.11.2024", "ul. Parkovaya, d. 28", "12.16 p.3", 939},
{"S990RG77", "Ivanov I.I.", "Toyota Land Cruiser", "21.07.2024", "ul. Chekhova, d. 15", "12.5 p.3", 2990},
{"D370JF50", "Matveev A.I.", "Audi A6", "03.11.2024", "ul. Zhukova, d. 8", "12.5 p.2", 524},
{"K810RX23", "Mikhailov E.E.", "Volkswagen Polo", "29.07.2024", "ul. Shkolnaya, d. 19", "12.9 p.1", 3432 },
{"K079CD50", "Grishin S.G.", "Honda Accord", "22.02.2024", "ul. Tsvetochnaya, d. 17", "12.9 p.2", 1050},
{"J997UO50", "Matveev A.I.", "Chevrolet Cruze", "16.08.2024", "ul. Shkolnaya, d. 4", "12.19 p.3", 702},
{"K711DK23", "Popov I.I.", "Kia Rio", "29.11.2024", "ul. Solnechnaya, d. 8", "12.15 p.4", 1792},
{"T735YE50", "Vinogradov E.E.", "BMW X3", "28.08.2024", "ul. Molodezhnaya, d. 26", "12.9 p.2", 3446},
{"B597LW61", "Loginov O.N.", "Lada XRay", "13.03.2024", "ul. Gagarina, d. 27", "12.15 p.4", 3110},
```

Рисунок 1 – Демонстрационный набор данных

Хэш-функция Jenkins, эта функция используется для быстрого хеширования строк и создаёт достаточно хорошее распределение:

```
size_t HashFun(const std::string& str) {
    size_t hash_value = 0;
    for (char ch : str) {
        hash_value += static_cast<size_t>(ch);
        hash_value += (hash_value << 10);
        hash_value ^= (hash_value >> 6);
    }
    hash_value += (hash_value << 3);
    hash_value ^= (hash_value >> 11);
    hash_value += (hash_value << 15);
    return hash_value;
}
```

Для наглядности предоставим вид хеш-таблицы до ее рехеширования, а именно когда ее размер равен 11 (табл. 2).

Таблица 2 – Хеш-таблица размера 11 для демонстрационного набора данных

№	KEY	Opened	Deleted	prK	Hash % size
0	B456OR77	False	False	False	0
1	A123BC77	False	False	False	1
2	C789KN77	False	False	False	2
3	E654TR77	False	False	True	1
4	F987KR77	False	False	True	3
5	G432NC77	False	False	True	1
6	K987RP77	False	False	True	10
7		True	False	False	
8	D321OR99	False	False	False	8
9		True	False	False	
10	H654TM77	False	False	False	10

После рехеширования размер таблицы изменится до 23, а также позиции записей изменятся (табл. 3)

Таблица 3 – Хеш-таблица размера 23 для демонстрационного набора данных

№	KEY	Opened	Deleted	prK	Hash % size
0		True	False	True	
1	E654TR77	False	False	False	1
2	G432NC77	False	False	False	2
3	D321OR99	False	False	True	2
4		True	False	False	
5	B456OR77	False	False	False	5
6	A123BC77	False	False	False	6
7		True	False	False	
8		True	False	False	
9	H654TM77	False	False	False	9
10		True	False	False	
11		True	False	False	
12		True	False	False	
13		True	False	False	
14		True	False	False	
15	C789KN77	False	False	False	15
16		True	False	False	
17	K987RP77	False	False	False	17
18		True	False	False	
19	F987KR77	False	False	False	19
20		True	False	False	
21		True	False	False	
22		True	False	False	

2.3 Программная реализация

2.3.1 Код файла заголовка

```
#ifndef CODE_TASK_1_H
#define CODE_TASK_1_H

#include <string>
```

```cpp
#include <iostream>
#include <algorithm>
#include <iomanip>
#include <cmath>

struct Node {
    std::string key;

    bool prK = false;
    bool opened = true;
    bool deleted = false;

    int ind_data;
};


struct HashTable {
    size_t table_size = 5;
    size_t num_closed = 0;
    Node* hash_table;

    HashTable() {
        hash_table = new Node[table_size];
    }

    ~HashTable() {
        delete[] hash_table;
    }
};


void ResizeTable(HashTable&, float);
void PrintHashTable(HashTable&);
void InsertElem(HashTable&, std::string key, size_t ind_data);

bool isPrime(int num) {
    int q = (int)sqrt(num);

    if (q*q == num)
        return false;

    for (int i = 2; i < q; i++) {
        if (num % i == 0)
            return false;
    }

    return true;
}

size_t HashFun(const std::string& str) {
    size_t hash_value = 0;
    for (char ch : str) {
        hash_value += static_cast<size_t>(ch);
        hash_value += (hash_value << 10);
        hash_value ^= (hash_value >> 6);
    }
    hash_value += (hash_value << 3);
    hash_value ^= (hash_value >> 11);
    hash_value += (hash_value << 15);
```

```
    return hash_value;
}

void ResizeTable(HashTable& hashTable) {
    size_t old_size = hashTable.table_size;
    Node* old_table = hashTable.hash_table;

    size_t new_size = old_size * 2 + 1;
    while (!isPrime(new_size))
        new_size += 2;

    PrintHashTable(hashTable);

    hashTable.table_size = new_size;
    hashTable.hash_table = new Node[hashTable.table_size];
    hashTable.num_closed = 0;


    for (size_t i = 0; i < old_size; i++) {
        if (!old_table[i].opened) {
            InsertElem(hashTable, old_table[i].key, old_table[i].ind_data);
        }
    }

    delete[] old_table;
}

void InsertElem(HashTable& hashTable, std::string key, size_t ind_data) {
    size_t pos = HashFun(key) % hashTable.table_size;
    bool K = false;

    while (!hashTable.hash_table[pos].opened) {
        K = true;
        pos++;

        if (pos == hashTable.table_size) {
            pos = 0;
        }
    }

    if (!hashTable.hash_table[pos].deleted)
        hashTable.num_closed++;

    hashTable.hash_table[pos].key = key;
    hashTable.hash_table[pos].prK = K;
    hashTable.hash_table[pos].ind_data = ind_data;
    hashTable.hash_table[pos].opened = false;
    hashTable.hash_table[pos].deleted = false;

    if ((float)hashTable.num_closed / (float)hashTable.table_size > 0.75)
        ResizeTable(hashTable);
}

void DeleteElem(HashTable& hashTable, const std::string& key) {
    size_t pos = HashFun(key) % hashTable.table_size;

    while ((!hashTable.hash_table[pos].opened || hashTable.hash_table[pos].deleted)
            && hashTable.hash_table[pos].key != key) {
        pos++;
```

```
        if (pos == hashTable.table_size)
            pos = 0;
    }

    if (hashTable.hash_table[pos].key != key)
        return;

    hashTable.hash_table[pos].deleted = true;
    hashTable.hash_table[pos].opened = true;
}

void PrintHashTable(HashTable& table) {
    std::cout << std::right << std::setw(4) << "#" << std::setw(10) << "KEY"
        << std::setw(12) << "Opened" << std::setw(12) << "Deleted" << std::setw(8) <<
"prK" <<
        std::setw(16) << "Hash % size" << '\n';
    for (size_t i = 0; i < table.table_size; i++) {
        bool flag = !table.hash_table[i].opened && !table.hash_table[i].deleted;
        std::cout << std::right << std::setw(4) << i << std::setw(10) << (flag ?
table.hash_table[i].key : std::string(9, '.')) <<
                std::boolalpha << std::setw(12) << table.hash_table[i].opened <<
                std::setw(12) << table.hash_table[i].deleted <<
                std::setw(8) << table.hash_table[i].prK << std::setw(16) <<
                (flag ? std::to_string(HashFun(table.hash_table[i].key) %
table.table_size) : std::string(15, '.')) << '\n';
    }
}

size_t FindElem (HashTable& hashTable, const std::string& key) {
    size_t pos = HashFun(key) % hashTable.table_size;

    while ((!hashTable.hash_table[pos].opened || hashTable.hash_table[pos].deleted)
           && hashTable.hash_table[pos].key != key) {
        pos++;

        if (pos == hashTable.table_size)
            pos = 0;
    }

    if (hashTable.hash_table[pos].key != key || hashTable.hash_table[pos].deleted)
        return -1;

    return hashTable.hash_table[pos].ind_data;
}


#endif //CODE_TASK_1_H
```

2.3.2 Код основной программы

```
#include "task_1.h"
#include <vector>
#include <variant>

int main() {
    HashTable h;
```

```
    std::vector<std::vector<std::variant<std::string, int>>> data {
            {"A123BC77", "Ivanov I.I.", "Toyota Camry", "15.09.2024", "ul. Lenina, d.
12", "12.9 ch.2", 500},
            {"B456OR77", "Petrov P.P.", "Hyundai Solaris", "18.09.2024", "pr. Mira,
d. 8", "12.16 ch.3", 1000},
            {"C789KN77", "Smirnov A.A.", "Kia Rio", "20.09.2024", "ul. Pushkina, d.
5", "12.5 ch.1", 1500},
            {"D321OR99", "Sidorov V.V.", "Ford Focus", "12.09.2024", "ul. Gagarina,
d. 7", "12.8 ch.2", 2000},
            {"E654TR77", "Kuznetsov N.N.", "Nissan Qashqai", "14.09.2024", "pr.
Lomonosova, d. 22", "12.15 ch.4", 700},
            {"F987KR77", "Fedorov A.S.", "Mazda 3", "16.09.2024", "ul. Chaykovskogo,
d. 9", "12.19 ch.1", 1200},
            {"G432NC77", "Morozov D.M.", "Volkswagen Polo", "19.09.2024", "pr.
Sakharova, d. 4", "12.16 ch.1", 500},
            {"H654TM77", "Vasiliev I.V", "Skoda Octavia", "13.09.2024", "ul.
Tolstogo, d. 10", "12.9 ch.2", 2500},
            {"K987RP77", "Nikolaev S.S", "Renault Duster", "17.09.2024", "ul.
Chekhova, d. 15", "12.5 ch.2", 1800},
            {"L123BC77", "Zaitsev M.M.", "Lada Vesta", "21.09.2024", "ul. Gorkogo, d.
11", "12.6 ch.1", 900},
            {"N038EK34", "Korolev V.V.", "Mazda 6", "10.05.2024", "ul. Tolstogo, d.
15", "12.6 p.1", 2312},
            {"P743YA23", "Volkov S.A.", "Ford Explorer", "27.11.2024", "ul.
Parkovaya, d. 28", "12.16 p.3", 939},
            {"S990RG77", "Ivanov I.I.", "Toyota Land Cruiser", "21.07.2024", "ul.
Chekhova, d. 15", "12.5 p.3", 2990},
            {"D370JF50", "Matveev A.I.", "Audi A6", "03.11.2024", "ul. Zhukova, d.
8", "12.5 p.2", 524},
            {"K810RX23", "Mikhailov E.E.", "Volkswagen Polo", "29.07.2024", "ul.
Shkolnaya, d. 19", "12.9 p.1", 3432 },
            {"K079CD50", "Grishin S.G.", "Honda Accord", "22.02.2024", "ul.
Tsvetochnaya, d. 17", "12.9 p.2", 1050},
            {"J997UO50", "Matveev A.I.", "Chevrolet Cruze", "16.08.2024", "ul.
Shkolnaya, d. 4", "12.19 p.3", 702},
            {"K711DK23", "Popov I.I.", "Kia Rio", "29.11.2024", "ul. Solnechnaya, d.
8", "12.15 p.4", 1792},
            {"T735YE50", "Vinogradov E.E.", "BMW X3", "28.08.2024", "ul.
Molodezhnaya, d. 26", "12.9 p.2", 3446},
            {"B597LW61", "Loginov O.N.", "Lada XRay", "13.03.2024", "ul. Gagarina, d.
27", "12.15 p.4", 3110},
    };


    for (int i = 0; i < data.size(); i++) {
       InsertElem(h, std::visit([](auto&& arg) -> std::string {
            if constexpr (std::is_same_v<std::decay_t<decltype(arg)>, int>) {
                return std::to_string(arg);
            } else {
                return arg;
            }
        }, data[i][0]), i);
    }

    PrintHashTable(h);

    DeleteElem(h, "D321OR99");

    std::cout << '\n';
```

```
    PrintHashTable(h);
    std::cout << FindElem(h, "H654TM77") <<'\n' << '\n';

    DeleteElem(h, "C789KN77");
    PrintHashTable(h);

    std::cout << FindElem(h, "P743YA23") << '\n' << '\n'; // Поиск после удаленного
элемента
}
```

2.4 Тестирование рехеширования

Вид хеш-таблицы размера 11 до момента ее рехеширования показан на рис. 2.

#	KEY	OpenKey	DeletedKey	prK	Hash % size
0	C789KN77	false	false	true	3
1	D321OR99	false	false	true	4
2	.....	true	false	false	.....
3	B456OR77	false	false	false	3
4	A123BC77	false	false	false	4
#	KEY	OpenKey	DeletedKey	prK	Hash % size
0	B456OR77	false	false	false	0
1	A123BC77	false	false	false	1
2	C789KN77	false	false	false	2
3	E654TR77	false	false	true	1
4	F987KR77	false	false	true	3
5	G432NC77	false	false	true	1
6	K987RP77	false	false	true	10
7	.....	true	false	false	.....
8	D321OR99	false	false	false	8
9	.....	true	false	false	.....
10	H654TM77	false	false	false	10

Рисунок 2 – Вид хеш-таблицы до ее рехеширования

После рехеширования хеш-таблица приобретет следующий вид (рис. 3):

#	KEY	OpenKey	DeletedKey	prK	Hash % size
0	.....	true	false	false	.....
1	E654TR77	false	false	false	1
2	G432NC77	false	false	false	2
3	D321OR99	false	false	true	2
4	.....	true	false	false	.....
5	B456OR77	false	false	false	5
6	A123BC77	false	false	false	6
7	.....	true	false	false	.....
8	.....	true	false	false	.....
9	H654TM77	false	false	false	9
10	.....	true	false	false	.....
11	.....	true	false	false	.....
12	.....	true	false	false	.....
13	.....	true	false	false	.....
14	.....	true	false	false	.....
15	C789KN77	false	false	false	15
16	.....	true	false	false	.....
17	K987RP77	false	false	false	17
18	.....	true	false	false	.....
19	F987KR77	false	false	false	19
20	.....	true	false	false	.....
21	.....	true	false	false	.....
22	.....	true	false	false	.....

Рисунок 3 – Вид хеш-таблицы после ее рехеширования

ЗАДАНИЕ 2

2.1 Требования к выполнению задания

Программа должна управлять двумя структурами данных: файл, хеш-таблица. Управление структурами предусматривает определение самой структуры как хранилища данных, так и операций над ними. Для повышения структуризации кода программы, требуется использовать два файла заголовков для локализации этих структур и третий заголовочный файл, который будет содержать операции доступа к записям файла посредством хеш-таблицы.

Заголовочные файлы должны содержать определения структуры данных, прототипы функций, реализующих операции и реализацию операций.

1. Разработать программу.

2. Подключить файл заголовка, созданный во второй практической работе, по управлению двоичным файлом.

2.1. Заголовочный файл должен содержать:

− определение структуры записи (определена в варианте задания табл. XXX) двоичного файла;

− операции управления двоичным файлом: создание двоичного файла из текстового, добавление записи в двоичный файл, удаление записи с заданным ключом из файла, чтение записи файла по заданному номеру записи и возвращение значения записи.

2.2. Выполнить тестирование всех операций из функции main программы.

2.3. После корректного выполнения программой всех операций:

− создать в файле заголовка функцию с именем testBinFile;

− содержание функции main переместить в эту функцию;

− из функции main вызвать функцию testBinFile и убедиться, что программа корректно работает.

3. Подключить файл заголовка по управлению хеш-таблицей, созданный в этой практической работе в задании 1.

4. Создать в программе новый заголовочный файл управления файлом посредством хеш-таблицы.

4.1. Подключить в нем заголовочные файлы: управления хеш-таблицей, управления двоичным файлом.

4.2. Разработать и реализовать операции управления файлом посредством хеш-таблицы:

1) прочитать запись из файла и вставить элемент в таблицу (элемент включает: ключ и номер записи с этим ключом в файле);

2) удалить запись из таблицы при заданном значении ключа и соответственно удалить 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) и вернуть запись.

4.3. Выполнить тестирование и отладку операций по мере их реализации, а не все сразу. Возможно, придется внести изменения в код каких-то функций.

5. Выполнить тестирование программы для файла небольшого (не менее 100) и большого размера (до 1 000 000). по количеству записей.

5.1. Подготовить тесты для тестирования программы в текстовом файл, затем создайте бинарный файл. Достаточно только значений ключей.

5.2. Включить в файл записи, приводящие и не приводящие к коллизиям.

5.3. Определить размер таблицы так, чтобы потребовалось рехеширование.

5.4. Определите время поиска 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.

3.2 Ход решения

3.2.1 Структуры записи элемента файла

Для работы с записями об нарушениях используется следующая структура, которая имеет фиксированную длину, а именно, **178 байт.**

```
struct Violation {
    char carNumber[16];
    char name[32];
    char model[32];
    char data[32];
    char place[32];
    char article[32];
    short fine;
}
```

3.2.2 Структура хеш-таблицы

Дальше показана структура хеш-таблицы и ее записи, реализованные в прошлом задание, но используемые повторно в этом.

```
struct Node {
    std::string key;

    bool prK = false;
    bool opened = true;
    bool deleted = false;

    int ind_data;
};

struct HashTable {
    size_t table_size = 5;
    size_t num_closed = 0;
    Node* hash_table;

    HashTable() {
        hash_table = new Node[table_size];
    }

    ~HashTable() {
        delete[] hash_table;
    }
};
```

3.2.3 Прототипы операций по управления файлом посредством хеш-таблицы.

```
bool ExtractRecordToHash(HashTable& h, const std::string& s_bin_file, size_t
ordinal_num);

bool ChangeIndDataInTable (HashTable& h, const std::string& car_num, size_t ind_new)
```

```
bool DelRecord(HashTable& h, const std::string& s_bin_file, const std::string& key);

Violation GetViolationByKey(HashTable& h, const std::string& s_bin_file, const
std::string& key, bool& SUCCESS_CODE);
```

Первые две из представленных функций возвращают булево значение, которое указывает на успешность завершения работы функции. Последняя функция возвращает саму запись.

ExtractRecordToHash – реализует преобразование записи по порядковому номеру из бинарного файла в элемент хеш-таблицы.

ChangeIndDataInTable – вспомогательная функция, используемая для изменения ссылки на данные записи в бинарном файле после удаления.

DelRecord – Удаляет запись из хеш-таблицы и бинарного файла.

GetViolationByKey – Если в хеш-таблицы существует элемент с определенным ключом, то возвращает полноценную запись из бинарного файла.

3.2.4 Описание реализации функций

ExtractRecordToHash:

В функции выполняется извлечение информации о нарушении из файла по пути s_bin_file по заданному номер ordinal_num. Для этого используется функци GetViolationByOrdinalNum, реализованной ранее, возвращающая структуру (или объект) с информацией о нарушении, например, номер транспортного средства carNumber. В процессе извлечения в переменную flag записывается логическое значение, указывающее на успешность извлечения.

Если flag установился в true (то есть запись была успешно извлечена), то:

- В хеш-таблицу h добавляется запись, ассоциирующая car_name с ordinal_num с помощью функции InsertElem.
- Функция возвращает true, указывая на успешное добавление записи.

Иначе: возвращает false.

ChangeIndDataInTable:

Функция ChangeIndDataInTable обновляет значение индекса ind_data для записи с определённым номером автомобиля car_num в хеш-таблице h. Сначала вычисляется хеш позиции, по которой может находиться запись. Затем функция выполняет проверку, является ли ячейка в хеш-таблице занятой и соответствующей нужному ключу car_num. Если она закрыта, удалена и не совпадает, выполняется циклический сдвиг до совпадения. При нахождении правильной записи обновляется её индекс на ind_new, и функция возвращает true. Если запись не найдена, возвращается false.

DelRecord:

Функция DelRecord удаляет запись о нарушении, связанного с ключом key, из бинарного файла s_bin_file и обновляет хеш-таблицу h. Сначала вызывается DeleteViolationByCarNum, который удаляет запись в файле и возвращает индекс ind удалённой записи. Если ind равен -1, это означает, что удаление не удалось, и функция возвращает false.

Если запись удалена успешно, функция DeleteElem удаляет её из хеш-таблицы. Далее GetViolationByOrdinalNum пытается получить запись из файла по индексу ind и извлечь номер автомобиля car_num. Если это не удаётся, функция возвращает false.

Если номер автомобиля успешно извлечён, ChangeIndDataInTable обновляет индекс этой записи в хеш-таблице, и функция возвращает true, указывая на успешное завершение операции.

GetViolationByKey:

Функция GetViolationByKey ищет информацию о нарушении по заданному ключу key в хеш-таблице h и возвращает её, если она найдена. Сначала переменной SUCCESS_CODE` присваивается значение false, указывая, что поиск пока неудачен. Затем функция ищет позицию элемента с помощью FindElem, который возвращает target — индекс искомого элемента.

Если target равен -1, это означает, что элемент не найден, и функция возвращает пустой объект Violation.

Если элемент найден, функция извлекает и возвращает запись о нарушении по порядковому номеру target из файла s_bin_file, обновляя SUCCESS_CODE на true для индикации успешного выполнения.

Общий алгоритм поиска записи с заданным ключом в файле посредством хеш-таблицы:

Алгоритм поиска записи с заданным ключом в файле посредством хеш-таблицы состоит из следующих шагов:

1. Поиск записи в хеш-таблице: Сначала выполняется поиск позиции ключа в хеш-таблице. Это делается с помощью функции, которая находит элемент по заданному ключу, используя хеш-функцию для вычисления его позиции. Если ключ не найден, алгоритм завершает работу, так как соответствующая запись отсутствует.

2. Извлечение индекса из таблицы: Когда элемент найден в хеш-таблице, из него извлекается индекс записи в бинарном файле. Этот индекс указывает на порядковый номер записи в файле, соответствующий ключу.

3. Чтение записи из файла: С использованием извлечённого индекса производится доступ к файлу, где считывается запись по этому номеру. Чтение записи осуществляется специальной функцией (GetViolationByOrdinalNum), которая получает запись по её порядковому номеру, возвращая нужные данные, такие как номер автомобиля или другие сведения о нарушении

3.3 Программная реализация

3.3.1 main.cpp

```
int main() {
    std::cout << sizeof(Violation) << '\n';

    HashTable h;
    std::string s_bin_file = "BinTest";

    // Вставка из бинарника в хэш-таблицу
    TextToBin("Z:\\MIREA\\SIAOD2\\SIAOD-3-sem\\6_1\\code\\SmallTest", s_bin_file);
    size_t cur_pos = 0;
    while (ExtractRecordToHash(h, s_bin_file, cur_pos++));
```

```
    bool flag;
    Violation v = GetViolationByKey(h, s_bin_file, "D370JF50", flag);

    if (flag)
        std::cout << v.ToString() << '\n';
    else
        std::cout << "Error" << '\n';

    DelRecord(h, s_bin_file, "D370JF50");

    v = GetViolationByKey(h, s_bin_file, "D370JF50", flag);

    if (flag)
        std::cout << v.ToString() << '\n';
    else
        std::cout << "Error" << '\n';

    v = GetViolationByKey(h, s_bin_file, "I304D76", flag);
    if (flag)
        std::cout << v.ToString() << "\n";
    else
        std::cout << "Error" << '\n';
}
```

3.3.2 Worker.h

```
#ifndef CODE_WORKER_H
#define CODE_WORKER_H

#include "BinFileWorker.h"
#include "task_1.h"

bool ExtractRecordToHash(HashTable& h, const std::string& s_bin_file, size_t
ordinal_num);
bool DelRecord(HashTable& h, const std::string& s_bin_file, const std::string& key);
bool ChangeIndDataInTable (HashTable& h, const std::string& car_num, size_t ind_new);
Violation GetViolationByKey(HashTable& h, const std::string& s_bin_file, const
std::string& key, bool& SUCCESS_CODE);

bool ExtractRecordToHash(HashTable &h, const std::string &s_bin_file, size_t
ordinal_num) {
    bool flag;
    std::string car_name = GetViolationByOrdinalNum(s_bin_file, ordinal_num,
flag).carNumber;

    if (flag) {
        InsertElem(h, car_name, ordinal_num);
        return true;
    }

    return false;
    DelRecord(h, s_bin_file, "1");
}

bool DelRecord(HashTable& h, const std::string& s_bin_file, const std::string& key) {
    size_t ind = DeleteViolationByCarNum(s_bin_file, key);
```

```
    if (ind == -1)
        return false;

    DeleteElem(h, key);

    bool flag = false;
    std::string car_num = GetViolationByOrdinalNum(s_bin_file, ind, flag).carNumber;

    if (!flag)
        return false;

    return ChangeIndDataInTable(h, car_num, ind);
}

Violation GetViolationByKey(HashTable& h, const std::string& s_bin_file, const
std::string& key, bool& SUCCESS_CODE) {
    SUCCESS_CODE = false;
    size_t target = FindElem(h, key);

    if (target == -1)
        return {};

    return GetViolationByOrdinalNum(s_bin_file, target, SUCCESS_CODE);
}

bool ChangeIndDataInTable (HashTable& h, const std::string& car_num, size_t ind_new)
{
    size_t pos = HashFun(car_num) % h.table_size;

    while ((!h.hash_table[pos].opened || h.hash_table[pos].deleted)
           && h.hash_table[pos].key != car_num) {
        pos++;

        if (pos == h.table_size)
            pos = 0;
    }

    if (h.hash_table[pos].key != car_num)
        return false;

    h.hash_table[pos].ind_data = ind_new;
    return true;
}



#endif //CODE_WORKER_H
```

3.3.3 BinFileWorker.h

```
#ifndef CODE_BINFILEWORKER_H
#define CODE_BINFILEWORKER_H
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include <cstring>
#include <filesystem>
```

```
namespace fs = std::filesystem;

std::vector<std::string> Split(std::string s, const std::string& separator) {
    std::vector<std::string> tokens;
    size_t pos = 0;
    std::string token;

    while ((pos = s.find(separator)) != std::string::npos) {
        token = s.substr(0, pos);

        if (!token.empty())
            tokens.push_back(token);

        s.erase(0, pos + separator.length());
    }

    if (!s.empty())
        tokens.push_back(s);

    return tokens;
}

struct Violation {
    char carNumber[16];
    char name[32];
    char model[32];
    char data[32];
    char place[32];
    char article[32];
    short fine;

    void SetFieldsByStr(const std::string& s) {
        std::vector<std::string> tokens = Split(s, ";");

        if (tokens.size() != 7) {
            std::cerr << "Reading Error" << '\n';
            return;
        }

        strcpy(carNumber, tokens[0].c_str());
        strcpy(name, tokens[1].c_str());
        strcpy(model, tokens[2].c_str());
        strcpy(data, tokens[3].c_str());
        strcpy(place, tokens[4].c_str());
        strcpy(article, tokens[5].c_str());
        fine = std::stoi(tokens[6]);
    }

    std::string ToString() {
        std::string res;

        res = carNumber + std::string(";") + name + std::string(";") + model +
std::string(";") +
                data + std::string(";") + place + std::string(";") + article +
std::string(";") +
                std::to_string(fine) + std::string(";");

        return res;
```

```
    }
};

void TextToBin(const std::string& file_name, const std::string& bin_file_name) {
    std::ifstream file(file_name);
    std::ofstream bin_file(bin_file_name, std::ios::binary);

    if (!file.is_open() || !bin_file.is_open()) {
        std::cerr << "Error opening file" << '\n';
        return;
    }

    std::string s;
    Violation violation;

    while (std::getline(file, s)) {
        violation.SetFieldsByStr(s);
        bin_file.write((char*)&violation, sizeof(Violation));
    }

    file.close();
    bin_file.close();
}

void BinToText(const std::string& file_name, const std::string& bin_file_name) {
    std::ifstream bin_file(bin_file_name, std::ios::binary);
    std::ofstream file(file_name);

    if (!bin_file.is_open() || !file.is_open()) {
        std::cerr << "Error opening file" << '\n';
        return;
    }

    Violation violation;

    while (bin_file.read((char*)&violation, sizeof(Violation))) {
        file << violation.ToString() << '\n';
    }

    bin_file.close();
    file.close();
}

void PrintAllViolationInBin(const std::string& bin_file_name) {
    std::ifstream bin_file(bin_file_name, std::ios::binary);

    if (!bin_file.is_open()) {
        std::cerr << "Error opening file" << '\n';
        return;
    }

    Violation violation;

    while (bin_file.read((char*)&violation, sizeof(Violation))) {
        std::cout << violation.ToString() << '\n';
    }

    bin_file.close();
}
```

```
Violation GetViolationByCarNum(const std::string& bin_file_name, const std::string&
car_num, bool& SUCCESS_CODE) {
    std::ifstream bin_file(bin_file_name, std::ios::binary);
    SUCCESS_CODE = false;

    if (!bin_file.is_open()) {
        std::cerr << "Error opening file" << '\n';
        return {};
    }

    Violation violation;

    while (bin_file.read((char*)&violation, sizeof(Violation))) {
        if (strcmp(violation.carNumber, car_num.c_str()) == 0) {
            bin_file.close();
            SUCCESS_CODE = true;
            return violation;
        }
    }

    bin_file.close();
    return {};
}

// Теперь возвращает индекс записи, удаленной и измененной на последнюю запись
size_t DeleteViolationByCarNum(const std::string& bin_file_name, const std::string&
car_num) {
    std::fstream bin_file(bin_file_name, std::ios::binary | std::ios::in |
std::ios::out | std::ios::ate);
    if (!bin_file.is_open())
        return -1;

    size_t file_size = fs::file_size(bin_file_name);
    bin_file.seekg(0, std::ios::beg);

    int violations_count = file_size / sizeof(Violation);

    std::streampos last_record_pos = (violations_count - 1) * sizeof(Violation);
    Violation last_violation;
    bin_file.seekg(last_record_pos);
    bin_file.read((char*)&last_violation, sizeof(Violation));

    bool deleted = false;
    std::streampos cur_pos = std::ios::beg;
    Violation violation;

    size_t ind_del = -1;
    bin_file.seekp(std::ios::beg);
    while (cur_pos != file_size) {
        bin_file.read((char*)&violation, sizeof(Violation));

        if (strcmp(violation.carNumber, car_num.c_str()) == 0 && ind_del == -1) {
            bin_file.seekp(cur_pos);
            bin_file.write((char*)&last_violation, sizeof(Violation));
            ind_del = cur_pos / sizeof(Violation);
        }

        cur_pos = bin_file.tellg();
```

```
    }

    bin_file.close();
    fs::resize_file(bin_file_name, file_size - sizeof(Violation));
    return ind_del;
}

void SelectionByCarNum(const std::string& bin_file_name, const std::string&
new_file_name, const std::string& car_num) {
    std::ifstream bin_file(bin_file_name, std::ios::binary);
    std::ofstream new_bin_file(new_file_name, std::ios::binary);

    if (!bin_file.is_open()) {
        std::cerr << "Error opening file" << '\n';
        return;
    }

    Violation violation;

    while (bin_file.read((char*)&violation, sizeof(Violation))) {
        if (strcmp(violation.carNumber, car_num.c_str()) == 0)
            new_bin_file << violation.ToString() << '\n';
    }

    bin_file.close();
}

void DoubleFine(const std::string& bin_file_name, const std::string& start_date,
                const std::string& end_date, const std::string& article) {

    std::fstream bin_file(bin_file_name, std::ios::binary | std::ios::in |
std::ios::out);

    if (!bin_file.is_open()) {
        std::cerr << "Error opening file" << '\n';
        return;
    }

    Violation violation;

    while (bin_file.read((char*)&violation, sizeof(Violation))) {
        if (violation.data >= start_date && violation.data <= end_date &&
strcmp(violation.article, article.c_str()) == 0) {
            violation.fine *= 2;
            bin_file.seekp(-sizeof(Violation), std::ios::cur);
            bin_file.write((char *) &violation, sizeof(Violation));
        }
    }

    bin_file.close();
}

void PrintBinFileInHex(const std::string& bin_file_name) {
    std::ifstream bin_file(bin_file_name, std::ios::binary);

    if (!bin_file.is_open()) {
        std::cerr << "Error opening file" << '\n';
        return;
    }
```

```
    char c;
    int k = 0;
    while (bin_file.get(c)) {
        std::cout << std::hex << std::setw(2) << std::setfill('0')
                  << (static_cast<unsigned int>(c) & 0xFF)
                  << ' ';
        k++;

        if (k % 16 == 0)
            std::cout << '\n';
    }

    bin_file.close();
}

Violation GetViolationByOrdinalNum(const std::string& bin_file_name, const size_t
ordinal_num, bool& SUCCESS_CODE) {
    std::fstream bin_file(bin_file_name, std::ios::binary | std::ios::in |
std::ios::out | std::ios::ate);
    SUCCESS_CODE = false;

    if (!bin_file.is_open()) {
        return {};
    }

    size_t file_size = fs::file_size(bin_file_name);
    bin_file.seekg(0, std::ios::beg);
    int violations_count = file_size / sizeof(Violation);

    if (ordinal_num > violations_count - 1 || ordinal_num < 0) {
        return {};
    }

    std::streampos record_pos = ordinal_num * sizeof(Violation);
    bin_file.seekg(record_pos, std::ios::beg);

    Violation target_violation;
    bin_file.read((char*)&target_violation, sizeof(Violation));

    SUCCESS_CODE = true;
    return target_violation;
}

#endif //CODE_BINFILEWORKER_H
```

3.3.4 task_1.h

```
#ifndef CODE_TASK_1_H
#define CODE_TASK_1_H

#include <string>
#include <iostream>
#include <algorithm>
#include <iomanip>
#include <cmath>

struct Node {
```

```
    std::string key;

    bool prK = false;
    bool opened = true;
    bool deleted = false;

    int ind_data;
};

struct HashTable {
    size_t table_size = 5;
    size_t num_closed = 0;
    Node* hash_table;

    HashTable() {
        hash_table = new Node[table_size];
    }

    ~HashTable() {
        delete[] hash_table;
    }
};

void ResizeTable(HashTable&);
void PrintHashTable(HashTable&);
void InsertElem(HashTable&, std::string key, size_t ind_data);

bool isPrime(int num) {
    int q = (int)sqrt(num);

    if (q*q == num)
        return false;

    for (int i = 2; i < q; i++) {
        if (num % i == 0)
            return false;
    }

    return true;
}

size_t HashFun(const std::string& str) {
    size_t hash_value = 0;
    for (char ch : str) {
        hash_value += static_cast<size_t>(ch);
        hash_value += (hash_value << 10);
        hash_value ^= (hash_value >> 6);
    }
    hash_value += (hash_value << 3);
    hash_value ^= (hash_value >> 11);
    hash_value += (hash_value << 15);
    return hash_value;
}

void ResizeTable(HashTable& hashTable) {
    size_t old_size = hashTable.table_size;
    Node* old_table = hashTable.hash_table;

    size_t new_size = old_size * 2 + 1;
```

```
    while (!isPrime(new_size))
        new_size += 2;

    hashTable.table_size = new_size;
    hashTable.hash_table = new Node[hashTable.table_size];
    hashTable.num_closed = 0;


    for (size_t i = 0; i < old_size; i++) {
        if (!old_table[i].opened) {
            InsertElem(hashTable, old_table[i].key, old_table[i].ind_data);
        }
    }

    delete[] old_table;
}

void InsertElem(HashTable& hashTable, std::string key, size_t ind_data) {
    size_t pos = HashFun(key) % hashTable.table_size;
    bool K = false;

    while (!hashTable.hash_table[pos].opened) {
        K = true;
        pos++;

        if (pos == hashTable.table_size) {
            pos = 0;
        }
    }

    if (!hashTable.hash_table[pos].deleted)
        hashTable.num_closed++;

    hashTable.hash_table[pos].key = key;
    hashTable.hash_table[pos].prK = K;
    hashTable.hash_table[pos].ind_data = ind_data;
    hashTable.hash_table[pos].opened = false;
    hashTable.hash_table[pos].deleted = false;

    if ((float)hashTable.num_closed / (float)hashTable.table_size > 0.75)
        ResizeTable(hashTable);
}

void DeleteElem(HashTable& hashTable, const std::string& key) {
    size_t pos = HashFun(key) % hashTable.table_size;

    while ((!hashTable.hash_table[pos].opened || hashTable.hash_table[pos].deleted)
            && hashTable.hash_table[pos].key != key) {
        pos++;

        if (pos == hashTable.table_size)
            pos = 0;
    }

    if (hashTable.hash_table[pos].key != key)
        return;

    hashTable.hash_table[pos].deleted = true;
    hashTable.hash_table[pos].opened = true;
```

```
}

void PrintHashTable(HashTable& table) {
    std::cout << std::right << std::setw(4) << "#" << std::setw(10) << "KEY"
        << std::setw(12) << "OpenKey" << std::setw(12) << "DeletedKey" <<
std::setw(8) << "prK" <<
        std::setw(16) << "Hash % size" << '\n';
    for (size_t i = 0; i < table.table_size; i++) {
        bool flag = !table.hash_table[i].opened && !table.hash_table[i].deleted;
        std::cout << std::right << std::setw(4) << i << std::setw(10) << (flag ?
table.hash_table[i].key : std::string(9, '.')) <<
                  std::boolalpha << std::setw(12) << table.hash_table[i].opened <<
                  std::setw(12) << table.hash_table[i].deleted <<
                  std::setw(8) << table.hash_table[i].prK << std::setw(16) <<
                  (flag ? std::to_string(HashFun(table.hash_table[i].key) %
table.table_size) : std::string(15, '.')) << '\n';
    }
}

size_t FindElem (HashTable& hashTable, const std::string& key) {
    size_t pos = HashFun(key) % hashTable.table_size;

    while ((!hashTable.hash_table[pos].opened || hashTable.hash_table[pos].deleted)
           && hashTable.hash_table[pos].key != key) {
        pos++;

        if (pos == hashTable.table_size)
            pos = 0;
    }

    if (hashTable.hash_table[pos].key != key || hashTable.hash_table[pos].deleted)
        return -1;

    return hashTable.hash_table[pos].ind_data;
}

#endif //CODE_TASK_1_H
```

3.4 Тестирование

Вывод после выполнения программы, описанной выше, представлен на рис. 4. В первой строке указан размер записи в байтах, во второй - результат поиска существующей записи, в третьей строке – результат поиска удаленной записи, в четвертой – результат поиска несуществующей записи.

```
178
D370JF50;Popov I.I.;BMW X5;28.09.2024;ul. Sovetskaya, d. 15;19.3 p.2;700;
Error
Error

Process finished with exit code 0
```

Рисунок 4 – Тестирование основной программы

Кроме того, необходимо получить время доступа к различным записям. Для этого воспользуемся следующим кодом:

```
auto start = std::chrono::high_resolution_clock::now();
Violation v = GetViolationByKey(h, s_bin_file, "D370JF50", flag);
auto end = std::chrono::high_resolution_clock::now();

std::chrono::duration<double> duration = end - start;
std::cout << "lead time: " << duration.count() << " sec" << std::endl;
```

Так, для поиска записи из начала файла (файл на 100000 записей) требуется:

lead time: 0.0001477 sec

Для записи из середины файла:

lead time: 0.000146 sec

Для записи из конца файла:

lead time: 0.0001355 sec

Следовательно, время поиска для всех трех случаев приблизительно совпадают.

Также необходимо дополнительно протестировать работоспособность программы при удалении какой-либо записи, ведь удаление происходит путем замены этой записи на последнюю. Для это воспользуемся кодом ниже:

```
GetViolationByKey(h, s_bin_file, "K079CD50", flag);
std::cout << std::boolalpha << flag << '\n'; //true

DelRecord(h, s_bin_file, "K079CD50");

GetViolationByKey(h, s_bin_file, "K079CD50", flag);
std::cout << std::boolalpha << flag << '\n'; //false

GetViolationByKey(h, s_bin_file, "B119GN23", flag); // Изначально последняя запись,
которая теперь должна заменить удаленную
std::cout << std::boolalpha << flag << '\n'; //true
std::cout << v.ToString() << '\n'; //B119GN23;Popov A.A.;Mercedes-Benz
E200;09.09.2024;pr. Mira, d. 5;12.8 p.2;300;
```

Результат выполнения:

```
true
false
true
B119GN23;Popov A.A.;Mercedes-Benz E200;09.09.2024;pr. Mira, d. 5;12.8 p.2;300;

Process finished with exit code 0
```

Рисунок 5 – Тестирование с удалением записи

ВЫВОД

Получили навыки по разработке программы управления хеш-таблицей и её адаптации для поиска данных в других структурах данных (файлах).

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Рысин, М. Л. Введение в структуры и алгоритмы обработки данных : учебное пособие / М. Л. Рысин, М. В. Сартаков, М. Б. Туманова. — Москва : РТУ МИРЭА, 2022 — Часть 2 : Поиск в тексте. Нелинейные структуры данных. Кодирование информации. Алгоритмические стратегии — 2022. — 111 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/310826 (дата обращения: 10.09.2024).
2. ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения : межгосударственный стандарт : дата введения 1992-01- 01 / Федеральное агентство по техническому регулированию. – Изд. официальное. – Москва : Стандартинформ, 2010. – 23 с