



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
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
«Хеширование и организация быстрого поиска данных»
по дисциплине
«Структуры и алгоритмы обработки данных»

Выполнил студент группы *ИКБО-03-22*

Хохлинов Д.И.

Принял

Сорокин А.В.

Практическая
работа выполнена

«__»_____2023 г.

«Зачтено»

«__»_____2023 г.

Москва 2023

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ	3
2 РАЗРАБОТКА ПРИЛОЖЕНИЯ	4
2.1 Структура двоичного файла.....	4
2.2 Генерация двоичного файла.....	4
2.3 Функционал приложения	7
2.4 Код программы.....	11
2.4 Тестирование	19
2.5 Тестирование на большом объеме данных	24
3 ОТВЕТЫ НА ВОПРОСЫ.....	31
4 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	35

1 ПОСТАНОВКА ЗАДАЧИ

1. Разработать программу, которая использует хеш-таблицу для организации прямого доступа к записям двоичного файла, структура записи которого приведена в варианте. Разработайте класс хеш-таблицы и реализуйте методы для операций:
 - 1) Хеширование (согласно варианту 25, используется открытая адресация с линейным пробированием);
 - 2) Чтение записи из файла и вставка записи в таблицу (запись включает: ключ и номер записи с этим ключом в файле);
 - 3) Удаление записи из таблицы и, соответственно, из файла;
 - 4) Поиск записи с заданным ключом в файле, используя хеш-таблицу;
 - 5) Выполнение рехеширования.
2. Провести тестирование программы на небольших объемах данных, введенных вручную. Разработанные тесты должны покрывать случаи возникновения коллизий и рехеширования.
3. Заполнить файл большим объемом данных. Провести исследование времени чтения записей из начала, середины и конца файла.
4. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования. Включить ответы на следующие вопросы:
 - a. Расскажите о назначении хеш-функции.
 - b. Что такое коллизия?
 - c. Что такое «открытый адрес» по отношению к хеш-таблице?
 - d. Как в хеш-таблице с открытым адресом реализуется коллизия?
 - e. Какая проблема, может возникнуть после удаления элемента из хеш-таблицы с открытым адресом и как ее устранить?
 - f. Что определяет коэффициент нагрузки в хеш-таблице?
 - g. Что такое «первичный кластер» в таблице с открытым адресом?
 - h. Как реализуется двойное хеширование?

2 РАЗРАБОТКА ПРИЛОЖЕНИЯ

2.1 Структура двоичного файла

Для выполнения задания использовалась следующая структура:

```
struct record {  
    char phone[11];  
    char address[100];  
};
```

Количество памяти, занимаемое полями этой структуры:

- phone – 11 байт;
- address – 100 байт.

Итого один экземпляр занимает 111 байт.

Структура двоичного файла, используемого для хранения записей такого типа, изображена на рисунке 1.

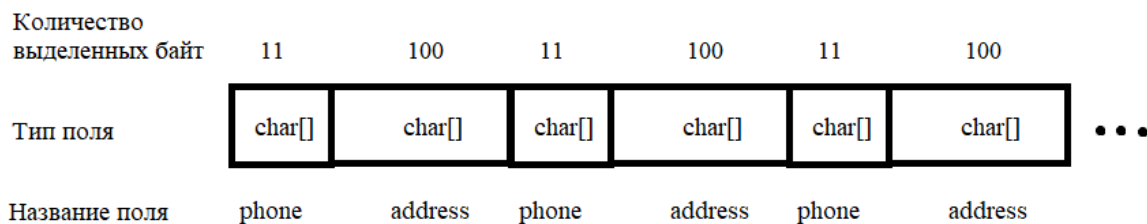


Рисунок 1 – Структура двоичного файла, используемого приложением

2.2 Генерация двоичного файла

Для создания описанного двоичного файла использовались следующие программы:

- Программа для транслитерации русского текста латиницей (реализована на языке Python, листинг 1):

Листинг 1 – Код программы для транслитерации

```
import argparse
import sys

def create_parser(): # command args handler
    parser = argparse.ArgumentParser()
    parser.add_argument('translit')
    parser.add_argument('input')
    parser.add_argument('output')

    return parser

if __name__ == '__main__':
    parser = create_parser()
    args = parser.parse_args(sys.argv[1:])
    translit_database_path = args.translit
    source_path = args.input
    output_path = args.output
    translit_database = []
    with open(translit_database_path, "r", encoding="utf-8") as translit_database_file:
        lines = translit_database_file.readlines()
        for line in lines:
            split = line.split()
            translit_database.append([split[0], split[1]])
    if len(translit_database) == 0:
        print("Ошибка при чтении файла с описанием замен. Процесс остановлен.")
    else:
        with open(source_path, "r", encoding="utf-8") as source:
            with open(output_path, "w", encoding="utf-8") as output:
                source_lines = source.readlines()
                for line in source_lines:
                    for translit in translit_database:
                        line = line.replace(translit[0], translit[1])
                    output.write(line)

    print("Процесс завершен.")
```

- Программа для генерации двоичного файла на основе текстового (реализована на языке C++, листинг 2):

Листинг 2 – Код программы для генерации двоичного файла

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

struct record {
```

Продолжение листинга 2

```
    char phone[11];
    char address[100];
};

int main()
{
    fstream input, output;
    string input_path;
    cin >> input_path;
    input = fstream(input_path, ios::in);
    output = fstream(input_path + "_bin.txt", ios::out |
ios::binary);
    record temp;
    while (!input.eof())
    {
        string temps;
        getline(input, temps);
        strcpy_s(temp.phone, temps.c_str());
        getline(input, temps);
        strcpy_s(temp.address, temps.c_str());
        output.write((char*)&temp, sizeof(record));
    }
    input.close();
    output.close();
}
```

В результате применения вышеописанных программ к текстовому файлу (листинг 3) был получен следующий двоичный файл (рисунок 2):

Листинг 3 – Исходный текстовый файл

```
4996008080
Moscow, Prospekt Vernadskogo, 78
3516492021
Челябинская область, город Истра, въезд Гоголя, 27
4991023923
Климентовский пер., 28, Москва, 162288
4912934834
Рязанская область, город Серебряные Пруды, въезд Ленина, 01
4920493923
Владимирская область, город Балашиха, ул. Славы, 48
3837461212
Новосибирская область, город Домодедово, шоссе Ленина, 90
3439992283
Свердловская область, город Подольск, наб. Бухарестская, 05
4751923845
Тамбовская область, город Павловский Посад, спуск 1905 года, 93
4739328456
Воронежская область, город Дмитров, въезд Домодедовская, 76
```


4) `record_number` – тип: `int`;

5) `key` – тип: массив из 11 элементов `char`;

b. Конструктор с параметром `hash`: присваивает соответствующему полю значение `hash`;

c. Деструктор: пустой.

4) Класс `hashtable`:

a. Поля:

1) `entries` – динамический массив типа указателей на `hashtable_entry`;

2) `used_entries` – тип: `int`;

b. Конструктор по умолчанию: пустой;

c. Конструктор с параметром `size`: создает `size` записей в поле `entries`;

d. Деструктор: очищает память, выделенную под массив `entries`;

e. Метод `getUsage`: возвращает результат деления значения поля `used_entries` на размер таблицы `table_size`;

f. Метод `show`: выводит на экран текущее состояние таблицы;

g. Метод `extend_table`: пока значение, возвращаемое методом `getUsage` таблицы, выше 0,75, увеличивает значение `table_size` в два раза. Затем добавляет новые записи в массив `entries` и вызывает метод `rehash` с параметрами `false, -1`;

h. Метод `rehash`. Принимает в качестве параметров режим работы `mode` и целое число `pos`. Открывает файл, с которым проводится работа; если его не удалось открыть, выводит соответствующее сообщение и завершает работу

приложения. В противном случае сохраняет имеющиеся данные в таблице по следующему алгоритму:

- 1) Объявить динамический целочисленный массив records;
- 2) Сохранить номер записи в очередной строке таблицы как `rn`;
- 3) Если `rn` не равно -1, то:
 1. Если `mode` – «ложь», то добавить `rn` в records.
Если `mode` – «истина», то:
 - a. Если `rn` равно `pos`, то ничего не происходит;
 - b. Если `rn` больше `pos`, то добавить `rn-1` в records;
 - c. Если `rn` меньше `pos`, то добавить `rn` в records.
 2. Очистить данные в строке таблицы.
- 4) Если конец таблицы не достигнут, то переход к п.2.

После сохранения данных производится сортировка records с помощью функции `std::sort`. Затем из файла считываются записи, указанные в массиве records, и записываются в таблицу с помощью `addData`.

- i. Метод `addData`. Принимает в качестве аргументов указатель `data` на структуру record и целое число `record_number`. Если переданная структура уже есть в таблице, выводит соответствующее сообщение. В противном случае вычисляет хэш `hash` переданной структуры. Если заполнено более 75% таблицы, вызывает метод `extend_table`. Объявляет

переменные $s = \text{TABLE_STEP} = 3$ (шаг пробирования) и $i = 0$. Затем пытается записать данные о структуре в строку таблицы с хэшем $\text{hash} + s * i$; если эта ячейка уже занята, помечает ее как ячейку, в которой наблюдалась коллизия и увеличивает i на 1, после чего повторяет попытку. В противном случае записывает данные в открытую ячейку, помечает ее как закрытую и увеличивает поле `used_entries` на 1;

- j. Метод `removeData`. Принимает в качестве аргумента указатель `data` на структуру `record`. Если переданная структура не найдена, выводит соответствующее сообщение. В противном случае перезаписывает файл так, чтобы остались все записи, кроме найденной, и вызывает метод `rehash` с параметрами `true`, `pos`, где `pos` – значение, возвращенное методом `findData` с аргументом `data->phone`;
- k. Метод `findData`. Принимает в качестве аргумента ключ `phone`. Вычисляет хэш `hash` структуры с ключом `phone`, объявляет счетчик итераций `iterations = 0`. Затем действует по следующему алгоритму:

- 1) Если ячейка с хэшем `hash` открыта, то:

- 1. Если в этой ячейке не возникало коллизий, то возвращает -1;
 - 2. В противном случае увеличивает `hash` на величину шага ($\text{TABLE_STEP} = 3$);

- 2) В противном случае:

1. Если ключ текущей ячейки совпадает с исходным, то возвращает номер записи, записанной в этой ячейке;
 2. В противном случае увеличивает hash на величину шага (TABLE_STEP = 3);
 - 3) Если hash больше или равен размеру таблицы, то уменьшает hash на table_size;
 - 4) Если количество итераций превысило двойной размер таблицы - аварийно завершить работу, так как это свидетельствует о возникших ошибках при работе с таблицей. Если нет, то переходит к п.1.
1. Меню модуля process. Подготавливает таблицу к работе (изначальный размер таблицы = table_size = 5). Затем в цикле предлагает пользователю одно из доступных действий. После выбора пользователем действия «выход» завершает работу.

2.4 Код программы

Запишем код программы на языке C++ (листинги 4-6):

Листинг 4 – Hashtable.h

```
#pragma once
#ifndef __hashtable_h__
#define __hashtable_h__
#include <iostream>
#include <algorithm>
#include <fstream>
#include <string>
#include <vector>
#include <conio.h>

using namespace std;

namespace Hashtable {
    struct record {
        char phone[11];
        char address[100];
        record();
        record(char* phone, char* address);
    };
};
```

Продолжение листинга 4

```
        bool operator == (record* right);
        bool operator != (record* right);
};

int getHash(record* data);

struct hashtable_entry {
    bool open = true;
    bool wasCollision = false;
    int hash = 0;
    int record_number = -1;
    char key[11];
    hashtable_entry(int hash);
    ~hashtable_entry();
};

class hashtable {
    vector<hashtable_entry*> entries;
    int used_entries = 0;
public:
    float getUsage();
    void show();
    bool addData(record* data, int record_number);
    bool removeData(record* data);
    int findData(char* phone);
    void process();
    void rehash(bool mode, int pos);
    void extend_table();
    hashtable();
    hashtable(int size);
    ~hashtable();
};
}
#endif
```

Листинг 5 – Hashtable.cpp

```
#include "Hashtable.h"

int table_size = 5;
const int TABLE_STEP = 3;
string path;
bool errors = false;

Hashtable::record::record()
{
    for (int i = 0; i < 11; i++)
    {
        this->phone[i] = 0;
        this->address[i] = 0;
    }
    for (int i = 11; i < 100; i++)
    {
        this->address[i] = 0;
    }
}
```

Продолжение листинга 5

```
    }
    strcpy_s(this->phone, "1234567890");
    strcpy_s(this->address, "Generic address");
}

Hashtable::record::record(char* phone, char* address)
{
    for (int i = 0; i < 11; i++)
    {
        this->phone[i] = 0;
        this->address[i] = 0;
    }
    for (int i = 11; i < 100; i++)
    {
        this->address[i] = 0;
    }
    strcpy_s(this->phone, phone);
    strcpy_s(this->address, address);
}

bool Hashtable::record::operator==(record* right)
{
    if (strcmp(this->phone, right->phone)) return false;
    if (strcmp(this->address, right->address)) return false;
    return true;
}

bool Hashtable::record::operator!=(record* right)
{
    return !(*this == right);
}

int Hashtable::getHash(record* data)
{
    unsigned int temp = 0;
    for (int i = 0; i < 11; i++)
    {
        temp += pow((abs((int)(data->phone[i])-48)), (i+1));
    }
    return temp % table_size;
}

Hashtable::hashtable_entry::hashtable_entry(int hash) : hash(hash)
{
    for (int i = 0; i < 11; i++)
    {
        this->key[i] = 0;
    }
}

Hashtable::hashtable_entry::~hashtable_entry()
{
}
```

Продолжение листинга 5

```
Hashtable::hashtable::hashtable()
{
}

Hashtable::hashtable::hashtable(int size)
{
    for (int i = 0; i < size; i++)
    {
        hashtable_entry* temp = new hashtable_entry(i);
        this->entries.push_back(temp);
    }
}

Hashtable::hashtable::~~hashtable()
{
    while (this->entries.size() > 0)
        this->entries.pop_back();
}

float Hashtable::hashtable::getUsage()
{
    return float(this->used_entries) / table_size;
}

void Hashtable::hashtable::extend_table() {
    int current_size = table_size;
    while (this->getUsage() >= 0.75) table_size *= 2;
    for (int i = current_size; i < table_size; i++)
    {
        hashtable_entry* temp = new hashtable_entry(i);
        this->entries.push_back(temp);
    }
    this->rehash(false, -1);
}

void Hashtable::hashtable::rehash(bool mode, int pos)
{
    this->used_entries = 0;
    fstream datafile = fstream(path, ios::in | ios::binary);
    if (datafile)
    {
        record temp;
        int current_record = 0;
        vector<int> records;
        for (int i = 0; i < table_size; i++)
        {
            int rn = this->entries.at(i)->record_number;
            if (rn != -1)
            {
                if (mode)
                {
                    if (rn == pos) {}
                    else if (rn > pos)
                        records.push_back(rn-1);
                }
            }
        }
    }
}
```

Продолжение листинга 5

```
        else
            records.push_back(rn);
    }
    else
        records.push_back(rn);
    this->entries.at(i)->record_number = -1;
    this->entries.at(i)->open = true;
    this->entries.at(i)->wasCollision = false;
    strcpy_s(this->entries.at(i)->key, "");
    }
}
sort(records.begin(), records.end());
while (!datafile.eof())
{
    if (current_record == records.size()) break;
    int start = 0;
    if (current_record) start = records[current_record-1];
    int end = records[current_record];
    for (int i = start; i < end; i++)
        datafile.read((char*)&temp, sizeof(record));
    this->addData(&temp, records[current_record]);
    current_record++;
}
}
else {
    cout << "Ошибка при чтении файла." << endl;
    errors = true;
}
}

bool Hashtable::hashtable::addData(record* data, int record_number)
{
    if (this->findData(data->phone) > 0)
    {
        cout << "Ошибка: эта запись уже есть в таблице." << endl;
        return false;
    }
    int hash = getHash(data);
    int c = TABLE_STEP;
    int i = 0;
    if (this->getUsage() >= 0.75) this->extend_table();
    while (this->entries.at(hash + c * i)->open == false)
    {
        this->entries.at(hash + c * i)->wasCollision = true;
        i++;
        if (hash + c * i >= table_size) hash -= table_size;
    }
    this->entries.at(hash + c * i)->record_number = record_number;
    this->entries.at(hash + c * i)->open = false;
    strcpy_s(this->entries.at(hash + c * i)->key, data->phone);
    this->used_entries++;
    return true;
}
```

Продолжение листинга 5

```
bool Hashtable::hashtable::removeData(record* data)
{
    int pos = this->findData(data->phone);
    if (pos < 0)
    {
        cout << "Элемент с ключом " << data->phone << " не найден в  
хэш-таблице" << endl;
        return false;
    }
    else
    {
        fstream tempfile("temp.txt", ios::out | ios::binary);
        fstream source(path, ios::in | ios::binary);
        record temp;
        int i = 1;
        while (source.read((char*)&temp, sizeof(record)))
        {
            if (i != pos)
            {
                tempfile.write((char*)&temp, sizeof(record));
            }
            i++;
        }
        source.close();
        tempfile.close();
        tempfile = fstream("temp.txt", ios::in | ios::binary);
        source = fstream(path, ios::out | ios::binary);
        while (tempfile.read((char*)&temp, sizeof(record)))
        {
            source.write((char*)&temp, sizeof(record));
        }
        source.close();
        tempfile.close();
        this->rehash(true, pos);
    }
    return true;
}

int Hashtable::hashtable::findData(char* phone)
{
    record data = record(phone, phone);
    int hash = getHash(&data);
    int iterations = 0;
    while (iterations < 2 * table_size) //для предотвращения  
зацикливания
    {
        if (this->entries.at(hash)->open)
        {
            if (this->entries.at(hash)->wasCollision == false)
            {
                return -1;
            }
        }
        else
        {

```


Продолжение листинга 5

```
        if (!strcmp(this->entries.at(hash)->key, data.phone))
        {
            return this->entries.at(hash)->record_number;
        }
    }
    hash += TABLE_STEP;
    if (hash >= table_size) hash -= table_size;
    iterations++;
}
cout << "Ошибка при поиске в хэш-таблице." << endl;
errors = true;
return -2;
}

void Hashtable::hashtable::show()
{
    for (int i = 0; i < this->entries.size(); i++)
    {
        hashtable_entry* temp = this->entries.at(i);
        cout << "hash: " << temp->hash << ", open: " << temp->open
        << ", was collision: " << temp->wasCollision << ", record number: " <<
        temp->record_number << ", key: " << temp->key << endl;
    }
}

void Hashtable::hashtable::process()
{
    path = "input.txt";
    fstream input;
    for (int i = 0; i < table_size; i++)
    {
        hashtable_entry* temp = new hashtable_entry(i);
        this->entries.push_back(temp);
    }
    record temp;
    int i = 1;
    bool running = true;
    char choose = ' ';
    int record_number, pos;
    char key[11];
    strcpy_s(key, "");
    while (running)
    {
        cout << "Выберите действие:" << endl
        << "[Q] - выход" << endl
        << "[A] - добавить запись в таблицу" << endl
        << "[R] - удалить запись из таблицы и из файла" <<
endl
        << "[F] - найти запись в таблице" << endl
        << "[S] - показать хэш-таблицу" << endl;
        choose = _getch();
        switch (tolower(choose))
        {
            case 'q':
                running = false;

```

Продолжение листинга 5

```
        break;
    case 's':
        cout << "Хэш-таблица (использовано полей: " << this->used_entries << "/" << table_size << "):" << endl;
        this->show();
        break;
    case 'a':
        cout << "Введите номер записи из файла, который необходимо вставить в таблицу: ";
        if (cin >> record_number)
        {
            input = fstream(path, ios::in | ios::binary);
            i = 1;
            while (input.read((char*)&temp, sizeof(record)))
            {
                if (i == record_number)
                    break;
                i++;
            }
            if (i != record_number)
            {
                cout << "Ошибка при чтении записи №" << record_number << endl;
            }
            else
            {
                this->addData(&temp, record_number);
            }
            input.close();
        }
        else
        {
            cout << "Ошибка при вводе значения." << endl;
        }
        break;
    case 'r':
        cout << "Введите номер телефона, запись с которым нужно удалить: ";
        cin >> key;
        if (this->removeData(&record(key, key)))
        {
            cout << "Запись с ключом " << key << " удалена успешно." << endl;
        }
        else
        {
            cout << "Ошибка при удалении записи с ключом " << key << endl;
        }
        break;
    case 'f':
        cout << "Введите номер телефона, запись с которым нужно найти: ";
        cin >> key;
        pos = this->findData(key);
```

Продолжение листинга 5

```
        if (pos > 0)
        {
            cout << "Запись с ключом " << key << " найдена на
позиции " << pos << endl;
        }
        else
        {
            cout << "Запись с ключом " << key << " не
найдена" << endl;
        }
        break;
    default:
        cout << "Неизвестное действие." << endl;
    }
    if (errors)
    {
        cout << "Были обнаружены ошибки, которые могут влиять
на работу программы. Процесс прекращен." << endl;
        running = false;
    }
    if (running)
    {
        system("pause");
        system("cls");
    }
}
}
```

Листинг 6 – main.cpp

```
#include <iostream>
#include <locale.h>
#include <fstream>
#include "Hashtable.h"
using namespace std;

int main()
{
    setlocale(LC_ALL, "ru");
    Hashtable::hashtable table = Hashtable::hashtable();
    table.process();
}
```

2.4 Тестирование

Проведем тестирование созданного приложения на небольшом двоичном файле (рисунок 2). Ход и результаты тестирования приведены на рисунках 3-13:

```
C:\Users\Дмитрий\source\repos\siaod_p2_3\Debug\siaod_p2_3.exe
Выберите действие:
[Q] - выход
[A] - добавить запись в таблицу
[R] - удалить запись из таблицы и из файла
[F] - найти запись в таблице
[S] - показать хэш-таблицу
Хэш-таблица (использовано полей: 0/5):
hash: 0, open: 1, was collision: 0, record number: -1, key:
hash: 1, open: 1, was collision: 0, record number: -1, key:
hash: 2, open: 1, was collision: 0, record number: -1, key:
hash: 3, open: 1, was collision: 0, record number: -1, key:
hash: 4, open: 1, was collision: 0, record number: -1, key:
Для продолжения нажмите любую клавишу . . .
```

Рисунок 3 – Хэш-таблица в стартовом состоянии

```
C:\Users\Дмитрий\source\repos\siaod_p2_3\Debug\siaod_p2_3.exe
Выберите действие:
[Q] - выход
[A] - добавить запись в таблицу
[R] - удалить запись из таблицы и из файла
[F] - найти запись в таблице
[S] - показать хэш-таблицу
Введите номер записи из файла, который необходимо вставить в таблицу: 40
Ошибка при чтении записи №40
Для продолжения нажмите любую клавишу . . .
```

Рисунок 4 – Попытка прочесть запись с несуществующим номером

```
C:\Users\Дмитрий\source\repos\siaod_p2_3\Debug\siaod_p2_3.exe
Выберите действие:
[Q] - выход
[A] - добавить запись в таблицу
[R] - удалить запись из таблицы и из файла
[F] - найти запись в таблице
[S] - показать хэш-таблицу
Хэш-таблица (использовано полей: 3/5):
hash: 0, open: 1, was collision: 0, record number: -1, key:
hash: 1, open: 0, was collision: 1, record number: 2, key: 3516492021
hash: 2, open: 1, was collision: 0, record number: -1, key:
hash: 3, open: 0, was collision: 1, record number: 1, key: 4996008080
hash: 4, open: 0, was collision: 0, record number: 3, key: 4991023923
Для продолжения нажмите любую клавишу . . .
```

Рисунок 5 – Хэш-таблица после чтения первых 3 записей

```
C:\Users\Дмитрий\source\repos\siaod_p2_3\Debug\siaod_p2_3.exe
Выберите действие:
[Q] - выход
[A] - добавить запись в таблицу
[R] - удалить запись из таблицы и из файла
[F] - найти запись в таблице
[S] - показать хэш-таблицу
Хэш-таблица (использовано полей: 4/5):
hash: 0, open: 1, was collision: 0, record number: -1, key:
hash: 1, open: 0, was collision: 1, record number: 2, key: 3516492021
hash: 2, open: 0, was collision: 0, record number: 4, key: 4912934834
hash: 3, open: 0, was collision: 1, record number: 1, key: 4996008080
hash: 4, open: 0, was collision: 1, record number: 3, key: 4991023923
Для продолжения нажмите любую клавишу . . .
```

Рисунок 6 – Хэш-таблица после чтения четвертой записи, при котором произошла коллизия

```
C:\Users\Дмитрий\source\repos\siaod_p2_3\Debug\siaod_p2_3.exe
Выберите действие:
[Q] - выход
[A] - добавить запись в таблицу
[R] - удалить запись из таблицы и из файла
[F] - найти запись в таблице
[S] - показать хэш-таблицу
Хэш-таблица (использовано полей: 5/10):
hash: 0, open: 1, was collision: 0, record number: -1, key:
hash: 1, open: 0, was collision: 0, record number: 2, key: 3516492021
hash: 2, open: 0, was collision: 0, record number: 5, key: 4920493923
hash: 3, open: 1, was collision: 0, record number: -1, key:
hash: 4, open: 1, was collision: 0, record number: -1, key:
hash: 5, open: 1, was collision: 0, record number: -1, key:
hash: 6, open: 0, was collision: 1, record number: 3, key: 4991023923
hash: 7, open: 1, was collision: 0, record number: -1, key:
hash: 8, open: 0, was collision: 1, record number: 1, key: 4996008080
hash: 9, open: 0, was collision: 0, record number: 4, key: 4912934834
Для продолжения нажмите любую клавишу . . .
```

Рисунок 7 – Хэш-таблица после чтения пятой записи. Произошло рехэширование из-за высокой загруженности таблицы (0.8)

C:\Users\Дмитрий\source\repos\siaod_p2_3\Debug\siaod_p2_3.exe

```
Выберите действие:
[Q] - выход
[A] - добавить запись в таблицу
[R] - удалить запись из таблицы и из файла
[F] - найти запись в таблице
[S] - показать хэш-таблицу
Хэш-таблица (использовано полей: 9/20):
hash: 0, open: 0, was collision: 0, record number: 9, key: 4739328456
hash: 1, open: 1, was collision: 0, record number: -1, key:
hash: 2, open: 1, was collision: 0, record number: -1, key:
hash: 3, open: 1, was collision: 0, record number: -1, key:
hash: 4, open: 1, was collision: 0, record number: -1, key:
hash: 5, open: 1, was collision: 0, record number: -1, key:
hash: 6, open: 1, was collision: 0, record number: -1, key:
hash: 7, open: 1, was collision: 0, record number: -1, key:
hash: 8, open: 0, was collision: 1, record number: 1, key: 4996008080
hash: 9, open: 1, was collision: 0, record number: -1, key:
hash: 10, open: 1, was collision: 0, record number: -1, key:
hash: 11, open: 0, was collision: 1, record number: 2, key: 3516492021
hash: 12, open: 0, was collision: 1, record number: 5, key: 4920493923
hash: 13, open: 1, was collision: 0, record number: -1, key:
hash: 14, open: 0, was collision: 0, record number: 8, key: 4751923845
hash: 15, open: 0, was collision: 1, record number: 6, key: 3837461212
hash: 16, open: 0, was collision: 1, record number: 3, key: 4991023923
hash: 17, open: 1, was collision: 0, record number: -1, key:
hash: 18, open: 0, was collision: 0, record number: 7, key: 3439992283
hash: 19, open: 0, was collision: 0, record number: 4, key: 4912934834
Для продолжения нажмите любую клавишу . . .
```

Рисунок 8 – Хэш-таблица после чтения 6-9 записей. При чтении 9 записи выполнялось рехэширование

```
C:\Users\Дмитрий\source\repos\siaod_p2_3\Debug\siaod_p2_3.exe
Выберите действие:
[Q] - выход
[A] - добавить запись в таблицу
[R] - удалить запись из таблицы и из файла
[F] - найти запись в таблице
[S] - показать хэш-таблицу
Введите номер телефона, запись с которым нужно удалить: 15
Элемент с ключом 15 не найден в хэш-таблице
Ошибка при удалении записи с ключом 15
Для продолжения нажмите любую клавишу . . .
```

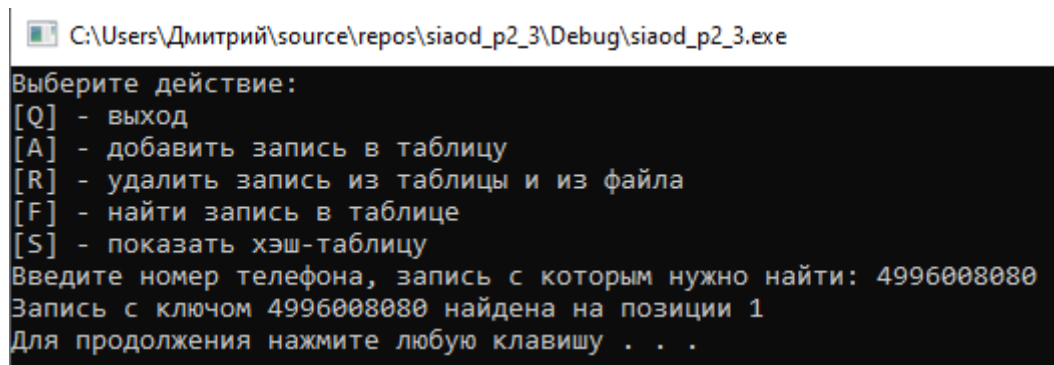
Рисунок 9 – Попытка удалить из таблицы несуществующую запись

```
C:\Users\Дмитрий\source\repos\siaod_p2_3\Debug\siaod_p2_3.exe
Выберите действие:
[Q] - выход
[A] - добавить запись в таблицу
[R] - удалить запись из таблицы и из файла
[F] - найти запись в таблице
[S] - показать хэш-таблицу
Введите номер телефона, запись с которым нужно удалить: 4751923845
Запись с ключом 4751923845 удалена успешно.
Для продолжения нажмите любую клавишу . . .
```

Рисунок 10 – Удаление из таблицы существующей записи

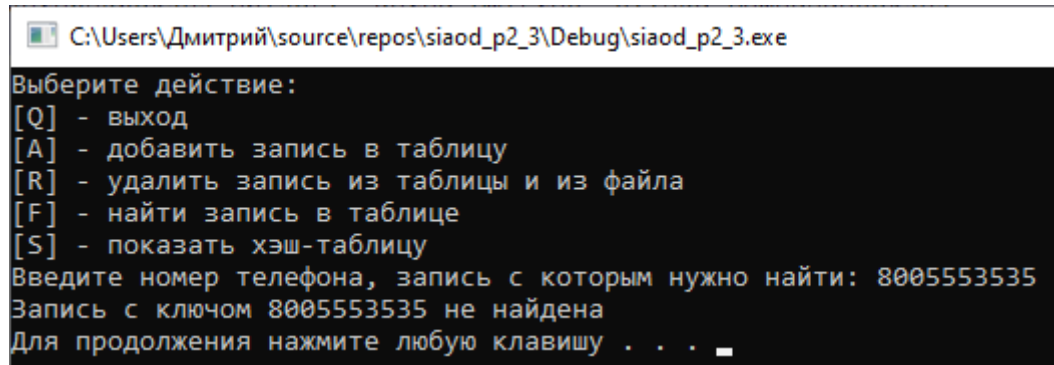
```
C:\Users\Дмитрий\source\repos\siaod_p2_3\Debug\siaod_p2_3.exe
Выберите действие:
[Q] - выход
[A] - добавить запись в таблицу
[R] - удалить запись из таблицы и из файла
[F] - найти запись в таблице
[S] - показать хэш-таблицу
Хэш-таблица (использовано полей: 8/20):
hash: 0, open: 0, was collision: 0, record number: 8, key: 4739328456
hash: 1, open: 1, was collision: 0, record number: -1, key:
hash: 2, open: 1, was collision: 0, record number: -1, key:
hash: 3, open: 1, was collision: 0, record number: -1, key:
hash: 4, open: 1, was collision: 0, record number: -1, key:
hash: 5, open: 1, was collision: 0, record number: -1, key:
hash: 6, open: 1, was collision: 0, record number: -1, key:
hash: 7, open: 1, was collision: 0, record number: -1, key:
hash: 8, open: 0, was collision: 1, record number: 1, key: 4996008080
hash: 9, open: 1, was collision: 0, record number: -1, key:
hash: 10, open: 1, was collision: 0, record number: -1, key:
hash: 11, open: 0, was collision: 0, record number: 2, key: 3516492021
hash: 12, open: 0, was collision: 1, record number: 5, key: 4920493923
hash: 13, open: 1, was collision: 0, record number: -1, key:
hash: 14, open: 1, was collision: 0, record number: -1, key:
hash: 15, open: 0, was collision: 1, record number: 6, key: 3837461212
hash: 16, open: 0, was collision: 1, record number: 3, key: 4991023923
hash: 17, open: 1, was collision: 0, record number: -1, key:
hash: 18, open: 0, was collision: 0, record number: 7, key: 3439992283
hash: 19, open: 0, was collision: 0, record number: 4, key: 4912934834
Для продолжения нажмите любую клавишу . . .
```

Рисунок 11 – Хэш-таблица после удаления, показанного на рисунке 10. Производилось рехэширование, связанное с изменением номеров записей



```
C:\Users\Дмитрий\source\repos\siaod_p2_3\Debug\siaod_p2_3.exe
Выберите действие:
[Q] - выход
[A] - добавить запись в таблицу
[R] - удалить запись из таблицы и из файла
[F] - найти запись в таблице
[S] - показать хэш-таблицу
Введите номер телефона, запись с которым нужно найти: 4996008080
Запись с ключом 4996008080 найдена на позиции 1
Для продолжения нажмите любую клавишу . . .
```

Рисунок 12 – Поиск существующей записи



```
C:\Users\Дмитрий\source\repos\siaod_p2_3\Debug\siaod_p2_3.exe
Выберите действие:
[Q] - выход
[A] - добавить запись в таблицу
[R] - удалить запись из таблицы и из файла
[F] - найти запись в таблице
[S] - показать хэш-таблицу
Введите номер телефона, запись с которым нужно найти: 8005553535
Запись с ключом 8005553535 не найдена
Для продолжения нажмите любую клавишу . . .
```

Рисунок 13 – Попытка поиска несуществующей записи

2.5 Тестирование на большом объеме данных

При тестировании работы программы с большим количеством данных использовался двоичный файл, полученный в результате манипуляций, описанных в пункте 2.2, над текстовым файлом большого объема (1000 записей).

Проведем модификацию используемой программы:

- Изменим метод process так, чтобы перед началом работы с таблицей она уже была заполнена всеми записями из файла;
- В методе process добавим новое доступное пользователю действие: прочитать запись, найденную по ключу, из файла и вывести ее на экран. При этом замеряется время поиска записи в хэш-таблице;
- Для уменьшения числа рехэширований установим начальный размер таблицы равным 1000.

Листинг 7 – Измененный метод process в файле Hashtable.cpp

```
void Hashtable::hashtable::process()
{
    path = "big input.txt";
    fstream input;
    for (int i = 0; i < table_size; i++)
    {
        hashtable_entry* temp = new hashtable_entry(i);
        this->entries.push_back(temp);
    }
    record temp;
    int i = 1;
    input = fstream(path, ios::in | ios::binary);
    while (input.read((char*)&temp, sizeof(record)))
    {
        this->addData(&temp, i);
        i++;
    }
    input.close();
    i = 1;
    auto start = chrono::steady_clock::now();
    auto end = chrono::steady_clock::now();
    bool running = true;
    char choose = ' ';
    int record_number, pos;
    char key[11];
    strcpy_s(key, "");
    while (running)
    {
        cout << "Выберите действие:" << endl
              << "[Q] - выход" << endl
              << "[A] - добавить запись в таблицу" << endl
              << "[R] - удалить запись из таблицы и из файла" <<
endl
              << "[F] - найти запись в таблице" << endl
              << "[O] - найти запись в таблице и вывести ее на
экран" << endl
              << "[S] - показать хэш-таблицу" << endl;
        choose = _getch();
        switch (tolower(choose))
        {
            case 'q':
                running = false;
                break;
            case 's':
                cout << "Хэш-таблица (использовано полей: " << this-
>used_entries << "/" << table_size << "):" << endl;
                this->show();
                break;
            case 'a':
                cout << "Введите номер записи из файла, который
необходимо вставить в таблицу: ";
                if (cin >> record_number)
                {
                    input = fstream(path, ios::in | ios::binary);
                    i = 1;
```

Продолжение листинга 7

```
        while (input.read((char*)&temp, sizeof(record)))
        {
            if (i == record_number)
                break;
            i++;
        }
        if (i != record_number)
        {
            cout << "Ошибка при чтении записи №" <<
record_number << endl;
        }
        else
        {
            this->addData(&temp, record_number);
        }
        input.close();
    }
    else
    {
        cout << "Ошибка при вводе значения." << endl;
    }
    break;
case 'r':
    cout << "Введите номер телефона, запись с которым
нужно удалить: ";
    cin >> key;
    if (this->removeData(&record(key, key)))
    {
        cout << "Запись с ключом " << key << " удалена
успешно." << endl;
    }
    else
    {
        cout << "Ошибка при удалении записи с ключом " <<
key << endl;
    }
    break;
case 'f':
    cout << "Введите номер телефона, запись с которым
нужно найти: ";
    cin >> key;
    pos = this->findData(key);
    if (pos > 0)
    {
        cout << "Запись с ключом " << key << " найдена на
позиции " << pos << endl;
    }
    else
    {
        cout << "Запись с ключом " << key << " не
найдена" << endl;
    }
    break;
case 'o':
```

Продолжение листинга 7

```
        cout << "Введите номер телефона, запись с которым  
нужно найти: ";  
        cin >> key;  
        start = chrono::steady_clock::now();  
        pos = this->findData(key);  
        end = chrono::steady_clock::now();  
        if (pos > 0)  
        {  
            cout << "Запись с ключом " << key << " найдена на  
позиции " << pos << endl;  
            input = fstream(path, ios::in | ios::binary);  
            i = 1;  
            while (input.read((char*)&temp, sizeof(record)))  
            {  
                if (i == pos)  
                    break;  
                i++;  
            }  
        }  
        else  
        {  
            cout << "Запись с ключом " << key << " не  
найдена" << endl;  
        }  
        if (pos > 0)  
        {  
            cout << "Найденная запись: " << temp.phone << "  
" << temp.address << endl;  
        }  
        cout << "Время поиска в хэш-таблице: " <<  
chrono::duration_cast<chrono::microseconds> (end - start).count() << "  
мкс" << endl;  
        break;  
    default:  
        cout << "Неизвестное действие." << endl;  
    }  
    if (errors)  
    {  
        cout << "Были обнаружены ошибки, которые могут влиять  
на работу программы. Процесс прекращен." << endl;  
        running = false;  
    }  
    if (running)  
    {  
        system("pause");  
        system("cls");  
    }  
}
```

Проведем тестирование модифицированной программы. В ходе этого тестирования будут произведены поиски записи в начале, середине и конце

файла, а также поиск несуществующей записи. Для уменьшения погрешности измерения тесты будут выполнены по 5 раз, а в качестве результата будет взято среднее значение. Результаты приведены в таблице 1, примеры вывода представлены на рисунках 14-17:

Таблица 1 – Время поиска различных записей с помощью хэш-таблицы

Искомый объект	Время поиска объекта, мкс					
	№ измерения					Среднее значение
	1	2	3	4	5	
Запись в начале файла (ключ: 4016588676)	6	7	7	6	7	6.6
Запись в середине файла (ключ: 2387214004)	9	7	9	10	10	9
Запись в конце файла (ключ: 4758479848)	15	15	15	15	16	15.2
Запись, не имеющаяся в файле (ключ: 8005553535)	15	15	16	15	15	15.2

```
C:\Users\Дмитрий\source\repos\siaod_p2_3\Debug\siaod_p2_3.exe
Выберите действие:
[Q] - выход
[A] - добавить запись в таблицу
[R] - удалить запись из таблицы и из файла
[F] - найти запись в таблице
[O] - найти запись в таблице и вывести ее на экран
[S] - показать хэш-таблицу
Введите номер телефона, запись с которым нужно найти: 4016588676
Запись с ключом 4016588676 найдена на позиции 13
Найденная запись: 4016588676; Leningradskaya oblast', gorod Lotoshino, ul. Pobedy, dom 53
Время поиска в хэш-таблице: 7 мкс
Для продолжения нажмите любую клавишу . . .
```

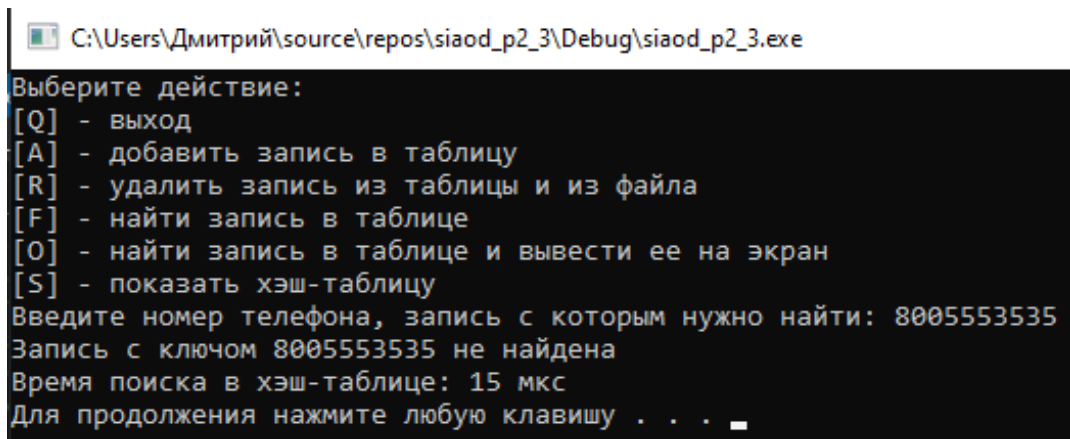
Рисунок 14 – Поиск записи в начале файла

```
C:\Users\Дмитрий\source\repos\siaod_p2_3\Debug\siaod_p2_3.exe
Выберите действие:
[Q] - выход
[A] - добавить запись в таблицу
[R] - удалить запись из таблицы и из файла
[F] - найти запись в таблице
[O] - найти запись в таблице и вывести ее на экран
[S] - показать хэш-таблицу
Введите номер телефона, запись с которым нужно найти: 2387214004
Запись с ключом 2387214004 найдена на позиции 565
Найденная запись: 2387214004; Permskii kraj, gorod Podol'sk, per. Lomonosova, dom 77
Время поиска в хэш-таблице: 10 мкс
Для продолжения нажмите любую клавишу . . .
```

Рисунок 15 – Поиск записи в середине файла

```
C:\Users\Дмитрий\source\repos\siaod_p2_3\Debug\siaod_p2_3.exe
Выберите действие:
[Q] - выход
[A] - добавить запись в таблицу
[R] - удалить запись из таблицы и из файла
[F] - найти запись в таблице
[O] - найти запись в таблице и вывести ее на экран
[S] - показать хэш-таблицу
Введите номер телефона, запись с которым нужно найти: 4758479848
Запись с ключом 4758479848 найдена на позиции 998
Найденная запись: 4758479848; Moskovskaya oblast', gorod Dorohovo, ul. Gagarina, dom 88
Время поиска в хэш-таблице: 15 мкс
Для продолжения нажмите любую клавишу . . .
```

Рисунок 16 – Поиск записи в конце файла



```
C:\Users\Дмитрий\source\repos\siaod_p2_3\Debug\siaod_p2_3.exe
Выберите действие:
[Q] - выход
[A] - добавить запись в таблицу
[R] - удалить запись из таблицы и из файла
[F] - найти запись в таблице
[O] - найти запись в таблице и вывести ее на экран
[S] - показать хэш-таблицу
Введите номер телефона, запись с которым нужно найти: 8005553535
Запись с ключом 8005553535 не найдена
Время поиска в хэш-таблице: 15 мкс
Для продолжения нажмите любую клавишу . . .
```

Рисунок 17 – Поиск несуществующей записи

По результатам тестирования видно, что время выполнения поиска в хэш-таблице отлично от $O(1)$, что связано с большим количеством коллизий в хэш-таблице.

3 ОТВЕТЫ НА ВОПРОСЫ

Вопрос 1 - Расскажите о назначении хэш-функции.

Хэш-функция – это математический алгоритм, преобразовывающий произвольные данные в число или строку фиксированной длины. При условии использования одного и того же алгоритма длина строки постоянна и не зависит от входных данных. Хэш-функции используются для шифрования паролей, создания хэш-таблиц, служащих для ускорения доступа к данным, и т.д.

Значение хэш-функции для одного и того же набора входных данных всегда одно и то же, для разных наборов входных данных значения хэш-функции могут совпадать.

Вопрос 2 - Что такое коллизия?

Коллизия – ситуация, при которой разным входным данным соответствует одно и то же значение хэш-функции. В общем случае коллизии неизбежны, но в оптимальных хэш-функциях их количество сведено к минимуму.

Вопрос 3 - Что такое «открытый адрес» по отношению к хеш-таблице?

«Открытый адрес» - строка хэш-таблицы, которая не занята никаким значением. Эта строка может быть перезаписана в процессе работы, тогда адрес станет «закрытым».

Вопрос 4 - Как в хеш-таблице с открытым адресом реализуется коллизия?

При возникновении коллизии в таблицах с открытым адресом происходит вычисление другого адреса внутри таблицы по некоторой функции, зависящей от текущего хэша и счетчика $i=0$, увеличивающемся на 1 при каждой коллизии. Таким образом, при попадании в «закрытую» ячейку происходит переход в другую, «связанную» с ней ячейку.

Вопрос 5 - Какая проблема, может возникнуть после удаления элемента из хэш-таблицы с открытым адресом и как ее устранить?

После удаления элемента из хэш-таблицы с открытым адресом может возникнуть ситуация, представленная на рисунке 18.

Открытая адресация с линейным пробированием (шаг = 3)
 $\text{hash}(\text{key1}) = \text{hash}(\text{key2}) = \text{hash}(\text{key3}) = 20$

Таблица до удаления элемента

хэш	ячейка открыта?	ключ	ссылка на значение
17	1		
18	1		
19	1		
20	0	key1	link1
21	1		
22	1		
23	0	key2	link2
24	1		
25	1		
26	0	key3	link3
27	1		
28	1		
29	1		
30	1		

Результат поиска - link3

Таблица после удаления элемента

хэш	ячейка открыта?	ключ	ссылка на значение
17	1		
18	1		
19	1		
20	0	key1	link1
21	1		
22	1		
23	1		
24	1		
25	1		
26	0	key3	link3
27	1		
28	1		
29	1		
30	1		

Результат поиска - записи с ключом key3 нет

Рисунок 18 – Возможная проблема при удалении элемента из хэш-таблицы

До удаления записи с ключом key2 поиск элемента с ключом key3 проводился по следующей схеме:

- 1) Вычислить хэш элемента (20);
- 2) Ячейка с хэшем 20 закрыта;
- 3) Ключ key1 не равен ключу key3 – искомый элемент не найден, выполнить смещение на 3 ячейки;
- 4) Ячейка с хэшем 23 закрыта;
- 5) Ключ key2 не равен ключу key3 – искомый элемент не найден, выполнить смещение на 3 ячейки;
- 6) Ячейка с хэшем 26 закрыта;

7) Ключ key3 равен ключу key3 – искомый элемент найден.

После удаления записи с ключом key2 поиск элемента с ключом key3 будет проходить так:

- 1) Вычислить хэш элемента (20);
- 2) Ячейка с хэшем 20 закрыта;
- 3) Ключ key1 не равен ключу key3 – искомый элемент не найден, выполнить смещение на 3 ячейки;
- 4) Ячейка с хэшем 23 открыта – искомый элемент не содержится в таблице.

То есть, получаем, что после удаления элемента key2 мы не сможем найти элемент key3 по таблице. Для решения этой проблемы нужно ввести дополнительное поле, показывающее, возникали ли коллизии при записи в эту ячейку (рисунок 19):

Открытая адресация с линейным пробированием (шаг = 3)
 $\text{hash}(\text{key1}) = \text{hash}(\text{key2}) = \text{hash}(\text{key3}) = 20$

Таблица до удаления элемента

хэш	ячейка открыта?	ключ	ссылка на значение	коллизия?
17	1			0
18	1			0
19	1			0
20	0	key1	link1	1
21	1			0
22	1			0
23	0	key2	link2	1
24	1			0
25	1			0
26	0	key3	link3	0
27	1			0
28	1			0
29	1			0
30	1			0

Таблица после удаления элемента

хэш	ячейка открыта?	ключ	ссылка на значение	коллизия?
17	1			0
18	1			0
19	1			0
20	0	key1	link1	1
21	1			0
22	1			0
23	1			1
24	1			0
25	1			0
26	0	key3	link3	0
27	1			0
28	1			0
29	1			0
30	1			0

Рисунок 19 – Хэш-таблица с дополнительным полем

Также нужно изменить алгоритм поиска: если очередная ячейка открыта, то нужно прочитать значение поля, отвечающего за коллизии: если в этой ячейке случались коллизии, то продолжаем поиск, так как искомый

элемент может быть расположен за удаленным элементом; если нет, то завершаем поиск.

Вопрос 6 - Что определяет коэффициент нагрузки в хэш-таблице?

Коэффициент нагрузки хэш-таблицы рассчитывается как $w = \frac{u}{s}$, где u – число записей в таблице, s – размер таблицы, и определяет степень загруженности хэш-таблицы. Если $w > 0.75$, то следует увеличить размер таблицы в 2 раза и провести рехеширование.

Вопрос 7 - Что такое «первичный кластер» в таблице с открытым адресом?

«Первичный кластер» - ситуация, при которой записи занимают соседние строки хэш-таблицы, образуя большое количество коллизий. Это может быть вызвано неудачным выбором хэш-функции или шага пробирования. Чтобы избежать данной ситуации, нужно подобрать такой шаг пробирования s , чтобы числа s и N – размер таблицы – были взаимно простыми, а s – не очень малым числом.

Вопрос 8 - Как реализуется двойное хеширование?

При реализации двойного хеширования используются две хэш-функции $f(x)$ и $h(x)$, которые имеют схожее, но не одинаковое действие. При этом функция $f(x)$ определяет хэш аргумента, $h(x)$ – смещение при возникновении коллизии. Эта схема позволяет уменьшить вероятность возникновения «первичного кластера» при заполнении таблицы.

4 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1) Задание на самостоятельную работу: https://online-edu.mirea.ru/pluginfile.php?file=%2F1144079%2Fmod_assign%2Fintr%2F0%2F%D0%A1%D0%B8%D0%90%D0%9E%D0%94%20%D0%A1%D0%B0%D0%BC%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%203%20%28%D1%85%D0%B5%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29.pdf&forcedownload=1, дата обращения: 05.10.23
- 2) Структуры и алгоритмы обработки данных – Лекция 2.1: https://online-edu.mirea.ru/pluginfile.php?file=%2F1126225%2Fmod_folder%2Fcontent%2F0%2F%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D1%8B%20%D0%B8%20%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B%20%D0%BE%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85_%D0%9B%D0%B5%D0%BA%D1%86%D0%B8%D1%8F_2.1.pptx&forcedownload=1, дата обращения: 05.10.23
- 3) Методические указания к практической работе: https://online-edu.mirea.ru/pluginfile.php?file=%2F1126230%2Fmod_folder%2Fcontent%2F0%2F%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D1%8B%20%D0%B8%20%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B%20%D0%BE%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85_%D0%9F%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F

[%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0_2.3%20%28%D0%A5%D0%B5%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29.docx&forcedownload=1](#), дата обращения: 05.10.23