|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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 № 2** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Хеширование и организация быстрого поиска данных»** | |
|  | |
| Выполнил студент группы ИКБО-02-21 | Семянников Н.С. |
| Принял преподаватель | Сорокин А.В. |

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

Москва 2022

# Постановка задачи

## Цель работы

Получить навыки по разработке хеш-таблиц и их применении.

## Постановка задачи

Разработайте приложение, которое использует хеш-таблицу для организации прямого доступа к записям файла, структура записи которого приведена в варианте №7.

Таблица - Задание варианта

|  |  |
| --- | --- |
| Метод хеширования (способ реализации коллизий) | Структура записи файла (ключ – подчеркнутое поле) |
| Цепное хеширование | Специализация вуза: код специальности, название вуза |

### Дано

Хеш-таблица для реализации коллизий по методу: цепное хеширование.

Файл двоичный с записями нефиксированной длины.

Структура записи файла: код специальности, название вуза

### Результат

Приложение, выполняющее следующие операции:

Управление хеш-таблицей: вставить ключ в таблицу, удалить ключ из таблицы, найти ключ в таблице, рехешировать таблицу

Управление файлом:

1) посредством хеш-таблицы: считать запись из файла, удалить запись из файла

2) добавить запись в файл.

## Вопросы

1) Расскажите о назначении хеш-функции.

2) Что такое коллизия?

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

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

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

6) Что определяет коэффициент нагрузки в хеш-таблице?

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

8) Как реализуется двойное хеширование?

## Ответы на вопросы

1. Хеш-функция — это математический алгоритм, который отображает данные произвольного размера в битовый массив фиксированного размера. Хэш-функция необходима для составления уникального ключа, чтобы в дальнейшем добавить запись в хеш-таблицу по данному ключу
2. Случай, при котором хеш-функция преобразует более чем один массив входных данных в одинаковые сводки, называется «коллизией».
3. Открытый адрес – это свободная ячейка хеш-таблицы, закрытый адрес – это занятая ячейка.
4. В случае коллизии, следующие ячейки проверяются линейно, пока не будет найдена пустая ячейка. А алгоритм поиска ищет ячейки в том же порядке, что и при вставке, пока не найдет нужный элемент или пустую ячейку, которая говорит о том, что ключ отсутствует.
5. При удалении элемента в хеш-таблица появляется «дыра», из-за которой поиск значений по данной хеш-таблице будет затруднён или и вовсе неосуществим. Чтобы хеш-таблица поддерживала удаление, требуется добавить параметр, говорящий, что запись удалена. Таким образом при удалении объект помечается как удалённый, а при добавлении нового элемента удалённый объект заменяется новым добавляемым элементом. При поиске, помимо равенства ключей, мы смотрим, удалён ли элемент, если да, то идём дальше.
6. Коэффициент нагрузки – количество хранимых элементов массива, деленное на число возможных значений хеш-таблицы. Когда количество записей в хэш-таблице превышает произведение коэффициента загрузки и текущей емкости, хеш-таблица автоматически увеличивается (рехешируется).
7. Первичный кластер — это явление, которое происходит, когда обработчик коллизий создает условие роста кластера. Из-за кластеризации увеличивается время выполнения операции добавления и поиска записи.
8. Двойное хеширование (англ. double hashing) — метод борьбы с коллизиями, возникающими при открытой адресации, основанный на использовании двух хеш-функций для построения различных последовательностей исследования хеш-таблицы.

# Решение

1. Хеш-таблица — класс
2. Структура элемента цепной хеш-таблицы: указатель на список, размер таблицы.
3. Структура элемента линейного двунаправленного списка: код специальности, название вуза, ссылка на предыдущий элемент списка, ссылка на следующий элемент списка. Новый элемент вставляется в начало списка.
4. Методы хеш-таблицы: вставить ключ в таблицу, удалить ключ из таблицы, найти ключ в таблице, рехешировать таблицу, вывод хеш-таблицы в консоль, записать хэш-таблицу в файл, вывести хэш-таблицу из файла.
5. Файл двоичный из записей нефиксированного размера.

**Структура записи файла:**

* код специальности — шестизначное число
* название вуза — строка неопределённой длины

**Операции по управлению файлом:**

* добавить запись в файл: занести запись в файл, вставить сведения о записи в хеш-таблицу
* удалить запись из файла: удалить из хеш-таблицы и из файла
* прочитать запись из указанной позиции (номеру/смещению)

# Алгоритмы операций

## Хеш-функция

int hashFunction(int x) {

return (x % BUCKET);

}

## Создание новой таблицы

void Hash::createNew(Hash& hash\_table, int size) {

int key;

string s[5] = { "МИРЭА", "МАИ", "МЭИ", "РЭУ", "МГУ" };

srand(time(0));

for (int i = 0; i < size / 2; i++)

{

key = rand() % 100000 + 100000;

insertItem(hash\_table, key, s[rand() % 5]);

}

}

## Вставка записи в таблицу

void Hash::insertItem(Hash& hash\_table, int key, string content) {

int index = hashFunction(key);

HashElement\* he = new HashElement(false, key, content);

list<HashElement>::iterator iter;

for (iter = table[index].begin(); iter != table[index].end(); iter++) {

if (((HashElement)\*iter).empty)

{

\*iter = \*he;

break;

}

}

if (iter == table[index].end()) {

table[index].push\_front(\*he);

}

count++;

if (count >= BUCKET \* 0.95)

{

rehash(hash\_table);

}

}

## Удаление записи из хеш-таблицы

void Hash::deleteItem(int key) {

int index = hashFunction(key);

for (list<HashElement>::iterator iter = table[index].begin(); iter != table[index].end(); iter++) {

if (((HashElement)\*iter).key == key) {

HashElement& x(\*iter);

x.empty = true;

count--;

writeFile();

cout << "Запись удалена";

break;

}

}

cout << "Запись не найдена";

}

## Поиск записи по ключу в таблице

HashElement Hash::find(int key) {

int index = hashFunction(key);

for (HashElement x : table[index]) {

if (x.key == key) {

return x;

}

}

return HashElement();

}

## Рехеширование

void Hash::rehash(Hash& old\_hash) {

BUCKET = BUCKET + 10;

Hash new\_hash(BUCKET);

for (int i = 0; i < BUCKET - 10; i++) {

for (HashElement x : table[i]) {

if (!x.empty) {

new\_hash.table[hashFunction(x.key)].push\_back(x);

new\_hash.count++;

}

}

}

old\_hash = new\_hash;

}

## Чтение из файла

void Hash::readFile(Hash& hash\_table) {

string str;

int key;

fstream f("university.bin", ios::in | ios::binary);

while (getline(f, str))

{

regex rgx("\\s+");

sregex\_token\_iterator iter(str.begin(), str.end(), rgx, -1);

key = stoi(\*iter);

insertItem(hash\_table, key, str.substr(to\_string(key).size() + 1));

}

f.close();

}

## Запись в файл

void Hash::writeFile() {

string str;

fstream f("university.bin", ios::out | ios::binary);

for (int i = 0; i < BUCKET; i++)

{

for (HashElement x : table[i])

{

if (!x.empty)

{

str = to\_string(x.key) + " " + x.name;

str += "\n";

f.write(str.c\_str(), size(str));

}

}

}

}

# Код

#include <iostream>

#include <list>

#include <regex>

#include <string>

#include <fstream>

#include <sstream>

using namespace std;

struct HashElement {

bool empty;

string name;

int key;

HashElement(bool empty, int key, string name) {

this->empty = empty;

this->key = key;

this->name = name;

}

HashElement() {}

};

class Hash {

int BUCKET;

int count = 0;

list <HashElement>\* table;

public:

Hash(int V);

void createNew(Hash& hash\_table, int size);

void readFile(Hash& hash\_table);

void writeFile();

void insertItem(Hash& hash\_table, int key, string content);

void deleteItem(int key);

int hashFunction(int x) {

return (x % BUCKET);

}

void rehash(Hash& old\_hash);

void displayHash();

HashElement find(int key);

};

Hash::Hash(int b) {

this->BUCKET = b;

table = new list <HashElement>[BUCKET];

}

void Hash::createNew(Hash& hash\_table, int size) {

int key;

string s[5] = { "МИРЭА", "МАИ", "МЭИ", "РЭУ", "МГУ" };

srand(time(0));

for (int i = 0; i < size / 2; i++)

{

key = rand() % 100000 + 100000;

insertItem(hash\_table, key, s[rand() % 5]);

}

}

//Считать из файла

void Hash::readFile(Hash& hash\_table) {

string str;

int key;

fstream f("university.bin", ios::in | ios::binary);

while (getline(f, str))

{

regex rgx("\\s+");

sregex\_token\_iterator iter(str.begin(), str.end(), rgx, -1);

key = stoi(\*iter);

insertItem(hash\_table, key, str.substr(to\_string(key).size() + 1));

}

f.close();

}

//Записать в файл

void Hash::writeFile() {

string str;

fstream f("university.bin", ios::out | ios::binary);

for (int i = 0; i < BUCKET; i++)

{

for (HashElement x : table[i])

{

if (!x.empty)

{

str = to\_string(x.key) + " " + x.name;

str += "\n";

f.write(str.c\_str(), size(str));

}

}

}

}

//Добавить элемент

void Hash::insertItem(Hash& hash\_table, int key, string content) {

int index = hashFunction(key);

HashElement\* he = new HashElement(false, key, content);

list<HashElement>::iterator iter;

for (iter = table[index].begin(); iter != table[index].end(); iter++) {

if (((HashElement)\*iter).empty)

{

\*iter = \*he;

break;

}

}

if (iter == table[index].end()) {

table[index].push\_front(\*he);

}

count++;

if (count >= BUCKET \* 0.95)

{

rehash(hash\_table);

}

}

//Удалить элемент

void Hash::deleteItem(int key) {

int index = hashFunction(key);

for (list<HashElement>::iterator iter = table[index].begin(); iter != table[index].end(); iter++) {

if (((HashElement)\*iter).key == key) {

HashElement& x(\*iter);

x.empty = true;

count--;

writeFile();

cout << "Запись удалена";

break;

}

}

cout << "Запись не найдена";

}

//Рехеширование

void Hash::rehash(Hash& old\_hash) {

BUCKET = BUCKET + 10;

Hash new\_hash(BUCKET);

for (int i = 0; i < BUCKET - 10; i++) {

for (HashElement x : table[i]) {

if (!x.empty) {

new\_hash.table[hashFunction(x.key)].push\_back(x);

new\_hash.count++;

}

}

}

old\_hash = new\_hash;

}

//Поиск элемента

HashElement Hash::find(int key) {

int index = hashFunction(key);

for (HashElement x : table[index]) {

if (x.key == key) {

return x;

}

}

return HashElement();

}

//Вывод таблицы

void Hash::displayHash() {

for (int i = 0; i < BUCKET; i++) {

cout << i;

for (HashElement x : table[i])

cout << " --> " << x.key << " " << x.name << " " << (x.empty == 1 ? "true" : "false");

cout << endl;

}

cout << endl;

}

int main() {

system("chcp 1251");

setlocale(0, "");

int a, key, size = 0;

string content, str;

Hash hash\_table(10);

while (true)

{

cout << "0 - Выход\n";

cout << "1 - Создать новую таблицу\n";

cout << "2 - Добавить элемент\n";

cout << "3 - Удалить элемент\n";

cout << "4 - Считать файл\n";

cout << "5 - Записать в файл\n";

cout << "6 - Найти запись\n";

cout << "7 - Рехешировать\n";

cout << "Выберите действие: ";

cin >> a;

switch (a)

{

case 1: {

cout << "Введите размер новой таблицы: ";

cin >> size;

hash\_table = Hash(10);

hash\_table.createNew(hash\_table, size);

hash\_table.displayHash();

break;

}

case 2: {

cout << "Введите код специализации (999999): ";

cin >> key;

if (key < 100000 || key > 999999)

{

cout << "Вы не справились с вводом кода специализации.\nДо свидания!";

exit(0);

}

cout << "Введите название ВУЗа: ";

cin >> content;

hash\_table.insertItem(hash\_table, key, content);

hash\_table.displayHash();

break;

}

case 3: {

hash\_table.displayHash();

cout << "Введите код специализации: ";

cin >> key;

hash\_table.deleteItem(key);

hash\_table.displayHash();

break;

}

case 4: {

hash\_table = Hash(10);

hash\_table.readFile(hash\_table);

hash\_table.displayHash();

break;

}

case 5: {

hash\_table.writeFile();

break;

}

case 6: {

cout << "Введите ключ для поиска: ";

cin >> key;

HashElement he = hash\_table.find(key);

if (he.key >= 0)

{

cout << "Ваша запись: " << he.key << " " << he.name << endl;

}

break;

}

case 7: {

hash\_table.rehash(hash\_table);

hash\_table.displayHash();

break;

}

default:

cout << "До свидания";

exit(0);

break;

}

}

return 0;

}

# Результат

## Меню

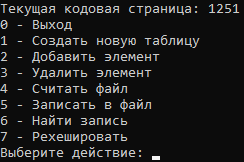


Рисунок - Меню

## Создание новой таблицы

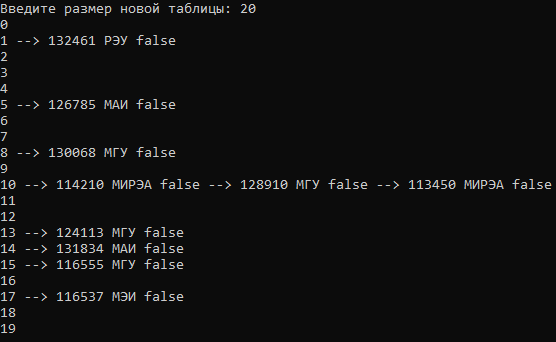


Рисунок - Создание таблицы

## Вставка записи в таблицу

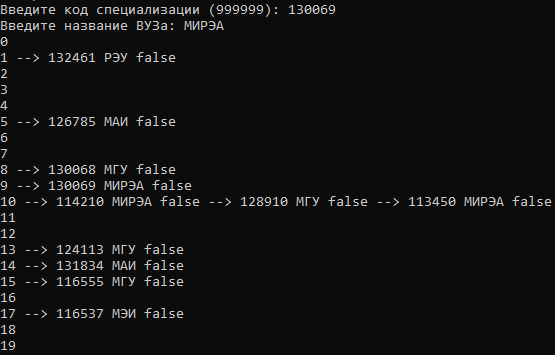


Рисунок - Добавление записи

## Удаление записи из таблицы

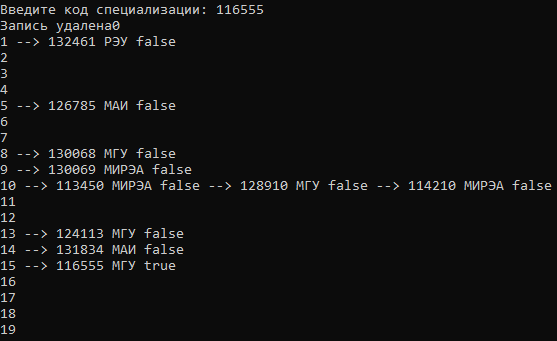


Рисунок - Удаление записи

## Поиск записи

|  |  |  |  |
| --- | --- | --- | --- |
| Количество элементов | Tначало(наносекунд) | Tсеред(наносекунд) | Тконец(наносекунд) |
| 100 | 7800 | 7900 | 8000 |
| 1000 | 8200 | 8300 | 8000 |

### 100 Элементов



Рисунок - Значение в начале



Рисунок - Значение в середине



Рисунок - Значение в конце

### 1000 Элементов



Рисунок - Значение в начале



Рисунок - Значение в середине



Рисунок - Значение в конце

## Рехеширование

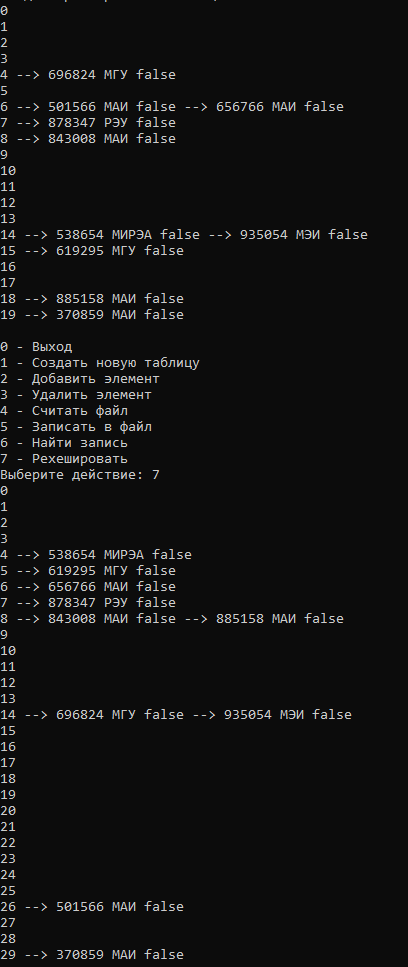


Рисунок – Рехеширование

## Чтение из файла

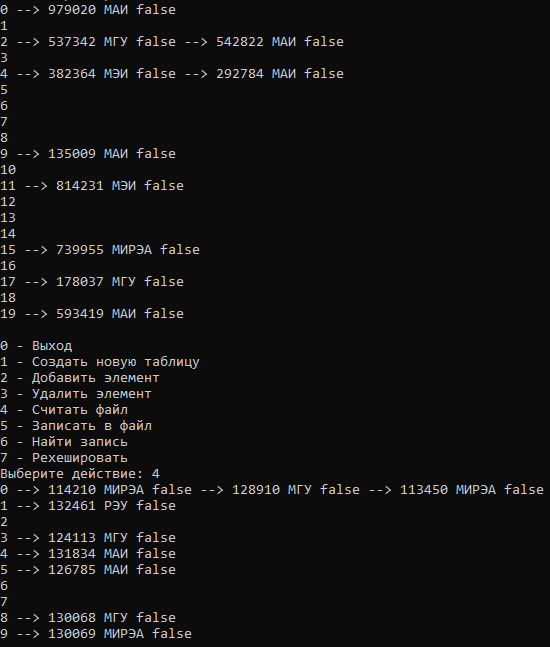


Рисунок - Чтение из файла

## Запись в файл

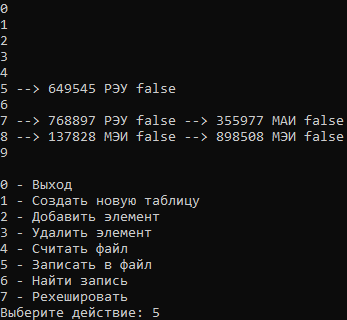


Рисунок - Записываем новую таблицу в файл

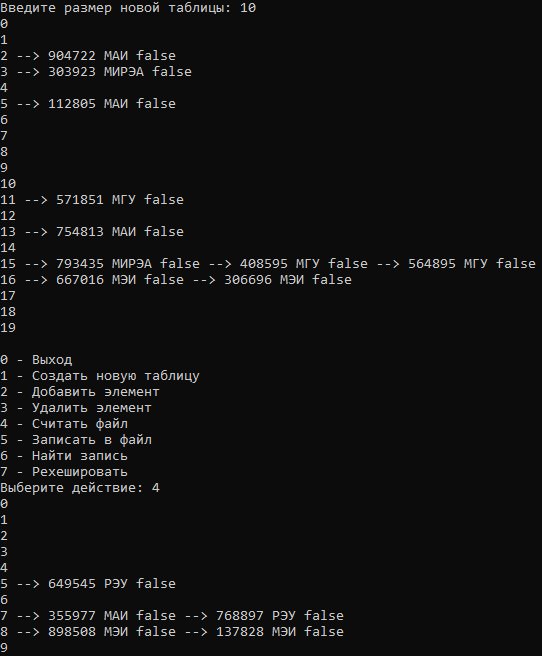


Рисунок - Создаём новую таблицу и считываем старую таблицу из файла

## Операции с файлом

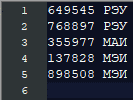


Рисунок Файл

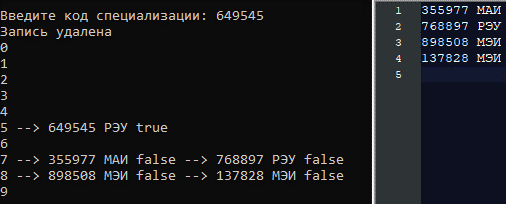


Рисунок - Удаление записи

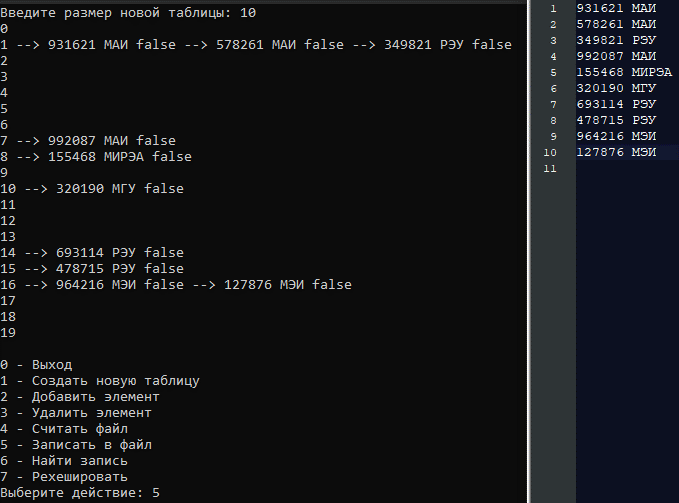


Рисунок - Запись новой таблицы в файл

# Вывод

Входе выполнения практической работы была изучена работа с хеш-таблицами, а также было изучено хеширование и организация быстрого поиска данных.

# Список литературы

1. Веб-ресурс про хэш-таблицы: <https://habr.com/ru/post/509220/> [дата обращения: 06.10.2022]
2. Система вопросов и ответов о программировании: <https://stackoverflow.com/>
3. Википедия: <https://ru.wikipedia.org/wiki/Хеш-функция> [дата обращения: 12.10.2022]