МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**«ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**Лабораторная работа:**

"Поиск данных с помощью хэш-таблиц"

Выполнила: студентка гр.РИС-23-3Б

Мокроусова Ангелина Андреевна

Проверила: доцент кафедры ИТАС

Полякова Ольга Андреевна

Пермь 2024

**Постановка задачи:**

1. Создать динамический массив из записей (в соответствии с вариантом), содержащий не менее 100 элементов. Для заполнения элементов массива использовать ДСЧ.

2. Предусмотреть сохранение массива в файл и загрузку массива из файла.

3. Предусмотреть возможность удаления элементов из массива (файла).

4. Выполнить поиск элемента в массиве по ключу в соответствии с вариантом. Для поиска использовать хэш-таблицу.

5. Подсчитать количество коллизий при размере хэш-таблицы 40, 75 и 90 элементов.

Данные: ФИО, дата рождения, адрес

Ключ: дата рождения

Хеш функция: H(k)=k mod M

Метод рехеширования: метод цепочек, метод открытой адресации

**Анализ задачи:**

1. Создание динамического массива:

- Необходимо создать структуру записи, содержащей ФИО, дату рождения и адрес.

- Создать массив из таких записей, заполняя его случайными значениями с использованием ДСЧ (Датчик случайных чисел).

2. Сохранение и загрузка массива из файла:

- Реализовать функцию сохранения массива в файл, записывая каждую запись поочередно.

- Реализовать функцию загрузки массива из файла, считывая каждую запись и добавляя ее в массив.

3. Удаление элементов:

- Удаление элементов: создать функцию, которая удалит запись по указанному ключу даты рождения.

4. Поиск элемента в массиве с использованием хэш-таблицы:

- Реализовать хэш-таблицу с заданным размером (например, 100 элементов).

- Для каждой записи в массиве вычислить хеш-значение по ключу даты рождения и добавить запись в таблицу.

- Для поиска элемента по ключу даты рождения использовать хеш-функцию и находить соответствующую запись в таблице.

5. Подсчет коллизий:

- Реализовать хеш-таблицу с различными размерами (40, 75, 90 элементов).

- Заполнить таблицу случайными данными, вычисляя для каждой записи хеш-значение и помещая ее в таблицу.

- Посчитать количество коллизий при различных размерах таблицы, используя метод цепочек или открытой адресации.

**Код на С++:**

#include <iostream>

#define \_USE\_MATH\_DEFINES

#include <string>

#include <ctime>

#include <fstream>

#include <math.h>

using namespace std;

const int M = 100;

const double A = M\_PI\_4;

int collisionCounter = 0;

struct Node {

string key;

string value;

Node\* next;

Node\* prev;

};

struct HashTable {

Node\* table[M];

int openAddressingCollisionCounter[M];

int chainingCollisionCounter;

HashTable() {

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

table[i] = nullptr;

openAddressingCollisionCounter[i] = 0;

}

chainingCollisionCounter = 0;

}

};

int getHash(double k) {

return static\_cast<int>(k) % M;

}

int getHash(string line) {

int n = 0;

for (int i = 0; i < line.size(); i++) {

n += static\_cast<int>(pow(line[i], 2) \* M\_2\_SQRTPI + abs(static\_cast<int>(line[i])) \* M\_SQRT1\_2);

}

return getHash(abs(n));

}

bool add(HashTable& table, string key, string elem) {

Node\* newNode = new Node;

newNode->key = key;

newNode->value = elem;

newNode->next = nullptr;

newNode->prev = nullptr;

int hash = getHash(key);

if (table.table[hash] == nullptr) {

table.table[hash] = newNode;

return true;

}

else {

Node\* current = table.table[hash];

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

newNode->prev = current;

table.chainingCollisionCounter++;

}

int i = 1;

while (table.table[(hash + i) % M] != nullptr) {

i++;

table.openAddressingCollisionCounter[hash]++;

}

table.table[(hash + i) % M] = newNode;

return true;

}

bool removeByKey(HashTable& table, string key) {

int hash = getHash(key);

Node\* current = table.table[hash];

while (current != nullptr) {

if (current->key == key) {

if (current->prev != nullptr) {

current->prev->next = current->next;

}

else {

table.table[hash] = current->next;

}

if (current->next != nullptr) {

current->next->prev = current->prev;

}

delete current;

table.chainingCollisionCounter--;

return true;

}

current = current->next;

}

return false;

}

Node\* get(HashTable& table, string key) {

int hash = getHash(key);

Node\* current = table.table[hash];

while (current != nullptr) {

if (current->key == key) {

return current;

}

current = current->next;

}

table.chainingCollisionCounter--;

return nullptr;

}

void print(HashTable& table) {

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

Node\* current = table.table[i];

while (current != nullptr) {

cout << "[" << current->key << ": " << current->value << "]\n";

current = current->next;

}

}

cout << endl;

}

string name[] = { "Иван", "Андрей", "Антон", "Амин", "Роберт", "Георгий", "Борис", "Владислав", "Дмитрий", "Никита" };

string patronymic[] = { "Сергеевич", "Ильич", "Григорьевич", "Артурович", "Денисович", "Алексеевич", "Александрович", "Валерьевич", "Дамирович", "Михайлович" };

string surname[] = { "Котов","Горбунов","Кудряшов","Быков","Зуев","Третьяков","Савельев","Панов","Рыбаков","Суворов" };

string strit[] = { "Центральная", "Молодежная", "Школьная", "Лесная", "Садовая", "Советская", "Новая", "Набережная", "Заречная", "Зеленая" };

string correctStr(int n, int length) {

string strn = to\_string(n);

while (strn.size() < length) {

strn = '0' + strn;

}

while (strn.size() > length) {

strn.erase(0, 1);

}

return strn;

}

string generateFullName() {

return surname[rand() % 10] + ' ' + name[rand() % 10] + ' ' + patronymic[rand() % 10];

}

string generateBirthday() {

return correctStr(rand() % 28 + 1, 2) + '.' + correctStr(rand() % 12 + 1, 2) + '.' + to\_string(rand() % 67 + 1960);

}

string generateAdres() {

return "Улица " + strit[rand() % 10] + "; Дом " + correctStr(rand() % 100, 2) + "; Квартира " + correctStr(rand() % 100, 2);

}

void write(HashTable& table) {

ofstream out;

out.open("File.txt");

if (out.is\_open()) {

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

Node\* current = table.table[i];

while (current != nullptr) {

out << current->key << endl << current->value << "\n";

current = current->next;

}

}

}

out.close();

}

void read(HashTable& table) {

ifstream in;

in.open("File.txt");

if (in.is\_open()) {

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

Node\* current = table.table[i];

while (current != nullptr) {

string str;

getline(in, str);

if (current == nullptr) {

current = new Node;

table.table[i] = current;

}

else {

current->next = new Node;

current->next->prev = current;

current = current->next;

}

current->key = str;

getline(in, str);

current->value = str;

current = current->next;

}

}

}

}

int main()

{

system("chcp 1251");

system("cls");

srand(time(NULL));

HashTable myTable;

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

string Birthday = generateBirthday();

string newHuman = generateFullName() + " | " + Birthday + " | " + generateAdres();

add(myTable, Birthday, newHuman);

}

cout << "Создана Хэш-Таблица: " << endl;

print(myTable);

string keyToRemove;

cout << "Введите ключ для удаления элемента (дату рождения): ";

cin >> keyToRemove;

cout << "Удаление по ключу: '" << keyToRemove << "':" << endl;

if (removeByKey(myTable, keyToRemove)) {

cout << "Элемент с ключом: '" << keyToRemove << "' успешно удален" << endl;

}

else {

cout << "Элемент с ключом: '" << keyToRemove << "' не найден" << endl;

}

print(myTable);

string keyToGet;

cout << "Введите ключ для поиска элемента (дату рождения): ";

cin >> keyToGet;

cout << "Получение элемента по ключу '" << keyToGet << "':" << endl;

Node\* node = get(myTable, keyToGet);

if (node != nullptr) {

cout << "Найден элемент: " << node->value << endl;

}

else {

cout << "Элемент с ключом '"

<< keyToGet << "' не найден." << endl;

}

cout << "\nчисло коллизий открытой адресации:\n";

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

cout << "хэш " << i << ": " << myTable.openAddressingCollisionCounter[i] << " коллизий\n";

}

cout << "\nчисло коллизий метода цепочек: " << myTable.chainingCollisionCounter << endl;

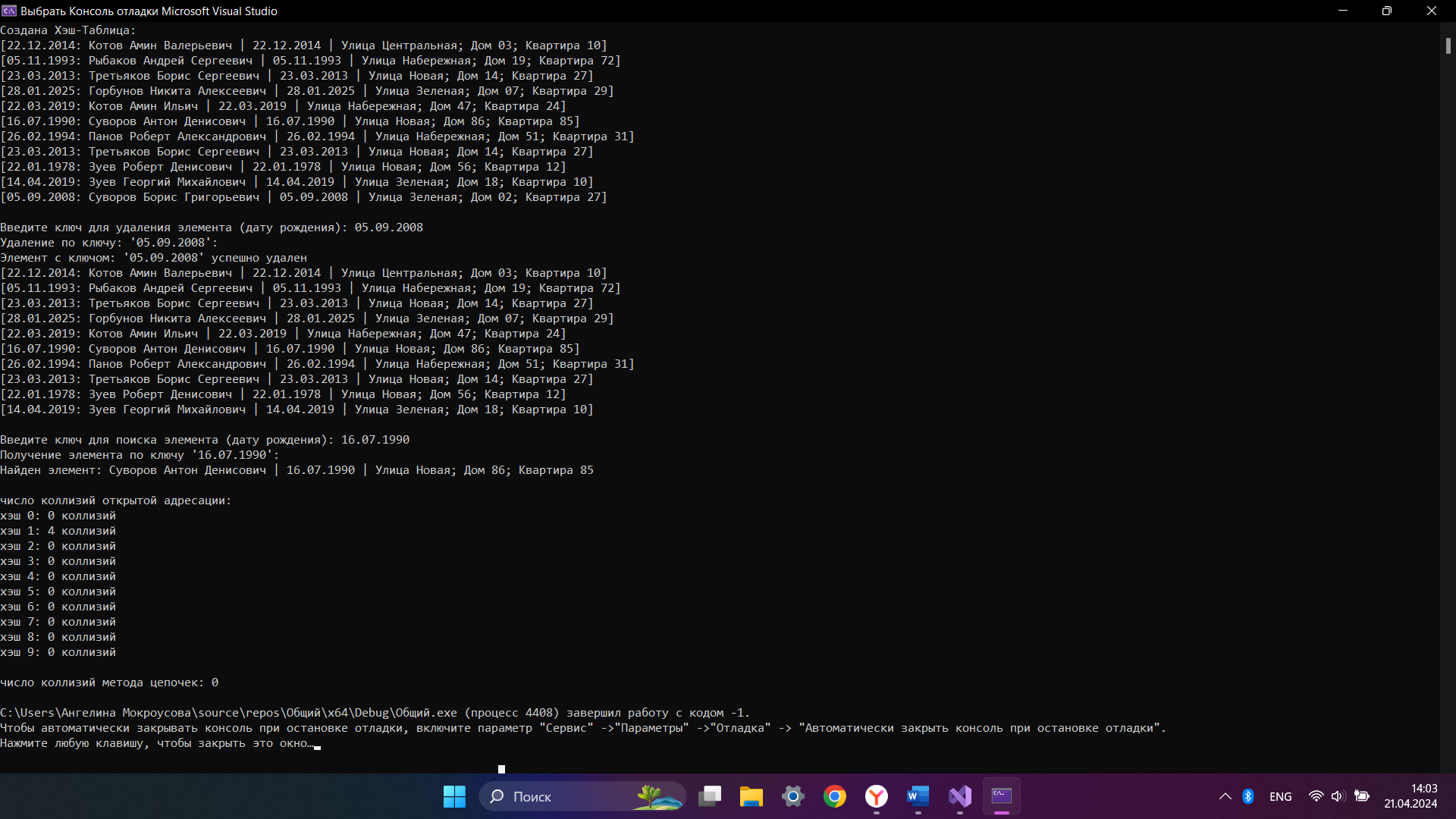
write(myTable);

read(myTable);

return 0;

}

**Скриншот работы программы:**



**Вывод:**

Программа упешно выполняет поставленную задачую.

Ответ:

Количесво получившихся коллизий на 40 элементов:

* Методом цепочек: 18
* Мотодом открытой адресации: 113

Количесво получившихся коллизий на 75 элементов:

* Методом цепочек: 38
* Мотодом открытой адресации: 291

Количесво получившихся коллизий на 90 элементов:

* Методом цепочек: 44
* Мотодом открытой адресации: 647

**Скриншот с GitHab:**

