Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа**

**«Хэш-таблицы»**

Выполнил:

студент группы РИС-23-2б

Ившин Максим Сергеевич

Проверила:

доцент кафедры ИТАС

О.А. Полякова

2024 г.

**Разработка алгоритма**

**Постановка задачи:**

Создать динамический массив из записей с полями: ФИО, дата рождения, номер телефона. Предусмотреть возможность добавления и удаления элементов из массива. Выполнить поиск элемента в массиве по ключу ФИО. Выполнить поиск элемента в массиве по ключу в соответствии с вариантом. Для поиска использовать хэш-таблицу. Подсчитать количество коллизий при размере хэш-таблицы 40, 75 и 90 элементов.

Хэш-функция: H(k)=k mod M;

**Анализ задачи:**

1. Создадим структуру Human в которой будут находится нужные поля
2. Также создадим структуру Hash\_Table, в которой будет находится динамическая структура vector, в которой будем реализовывать сам хэш-таблицу
3. Для заполнения таблицы создадим генераторы на основе rand(): для поля ФИО, номера телефона и даты рождения.
4. Функция хэширования будет создавать хэш индекс на основе входной строки «ФИО» и подстраивать под размер хэш-таблицы
5. Для устранения коллизии будем пользоваться двумя методами:методом отрытой адресации (поиск «свободного» места)и методом цепочек (использования массива массивов для реализации «цепочки»)

**Код на С++ для метода открытой адресации:**

#include <iostream>

#include <string>

#include <vector>

#include <cmath>

using namespace std;

const double A = 3.141592653589793; // иррациональная константа

int hashTable\_size;

extern int &M = hashTable\_size;

#include "Hash.h"

#include "Generators.h"

struct Human

{

    string nameSP = "-";

    string birthday = "-";

    string phoneNum = "-";

};

struct Hash\_Table

{

    vector<Human> table;

    int collision\_cnt = 0;

    void print()

    {

        cout << "Таблица: " << endl;

        int position = 1;

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

        {

            if (table[i].nameSP == "-")

                continue;

            cout << position << ")" << " ФИО: " << table[i].nameSP;

            cout << " | Дата рождения: " << table[i].birthday;

            cout << " | Номер телефона : " << table[i].phoneNum << endl;

            position++;

        }

    }

    void collision\_solve(int index, Human new\_element)

    {

        collision\_cnt++;

        int cnt = 0; // защита от зацикливания

        while (table[index].nameSP != "-" && cnt < M)

        {

            if (index == M)

            {

                index = -1;

            }

            index++;

            cnt++;

        }

        if (cnt == M)

            cout << endl

                 << "Ошибка устранения коллизии";

        else

            table[index] = new\_element;

    }

    int find\_element(string element)

    {

        int index = getHash(element);

        int cnt = 0; // защита от зацикливания

        while (table[index].nameSP != element && cnt <= M)

        {

            if (index == M)

                index = -1;

            index++;

            cnt++;

        }

        if (cnt >= M)

        {

            cout << endl << "Не удалось найти элемент" << endl;

            return -1;

        }

        else

        {

            cout << endl << "Элемент успешно найден, его номер: " << index + 1 << endl;

            return index;

        }

    }

    void add\_element()

    {

        Human temp\_human;

        cout << "Введите ФИО: ";

        cin.seekg(cin.eof());

        getline(cin, temp\_human.nameSP);

        cout << "Введите дату рождения: ";

        cin.seekg(cin.eof());

        getline(cin, temp\_human.birthday);

        cout << "Введите номер телефона: ";

        cin.seekg(cin.eof());

        getline(cin, temp\_human.phoneNum);

        int index = getHash(temp\_human.nameSP);

        if (table[index].nameSP != "-")

        {

            collision\_solve(index, temp\_human);

        }

        else

        {

            table[index] = temp\_human;

        }

    }

    void remove\_element(string element)

    {

        int index = getHash(element);

        int cnt = 0; // защита от зацикливания

        while (table[index].nameSP != element && cnt <= M)

        {

            if (index == M)

                index = -1;

            index++;

            cnt++;

        }

        if (cnt >= M)

            cout << endl

                 << "Не удалось найти элемент" << endl;

        else

        {

            table[index].nameSP = "-";

            table[index].phoneNum = "-";

            table[index].birthday = "-";

            cout << endl

                 << "Элемент успешно удален" << endl;

        }

    }

};

int main()

{

    srand(time(0));

    cout << "Кол-во человек: ";

    cin >> hashTable\_size;

    Hash\_Table people;

    people.table.resize(hashTable\_size);

    for (int i = 0; i < hashTable\_size; i++) // заполнение

    {

        Human temp\_human;

        temp\_human.nameSP = generate\_fullName();

        temp\_human.birthday = generate\_date();

        temp\_human.phoneNum = generate\_PhoneNum();

        int index = getHash(temp\_human.nameSP);

        if (people.table[index].nameSP != "-")

        {

            people.collision\_solve(index, temp\_human);

        }

        else

        {

            people.table[index] = temp\_human;

        }

    }

    cout << "Количество колллизий: " << people.collision\_cnt << endl;

    people.print();

    cout << "Введите ключ для удаления [ФИО]: ";

    string temp\_string;

    cin.seekg(cin.eof());

    getline(cin, temp\_string);

    people.remove\_element(temp\_string);

    people.print();

    cout << "\tДобавление элемента:" << endl;

    people.add\_element();

    people.print();

    cout << "Введите ключ для поиска [ФИО]: ";

    cin.seekg(cin.eof());

    getline(cin, temp\_string);

    people.find\_element(temp\_string);

}

**Код на С++ для метода цепочек:**

#include <iostream>

#include <string>

#include <vector>

#include <cmath>

using namespace std;

const double A = 3.141592653589793; // иррациональная константа

int hashTable\_size;

extern int &M = hashTable\_size;

#include "Hash.h"

#include "Generators.h"

struct Human

{

    string nameSP = "-";

    string birthday = "-";

    string phoneNum = "-";

};

struct Hash\_Table

{

    vector<vector<Human>> table;

    int collision\_cnt = 0;

    int size\_cnt = 0;

    void print()

    {

        cout << "Таблица: " << endl;

        int position = 1;

        for (int i = 0; i < size\_cnt; i++)

        {

            for (int j = 0; j < table[i].size(); j++)

            {

                cout << position << ")" << " ФИО: " << table[i][j].nameSP;

                cout << " | Дата рождения: " << table[i][j].birthday;

                cout << " | Номер телефона : " << table[i][j].phoneNum << endl;

                position++;

            }

        }

    }

    void collision\_solve(int index, Human new\_element)

    {

        collision\_cnt++;

        table[index].push\_back(new\_element);

    }

    int find\_element(string element)

    {

        int position = 1;

        int index = getHash(element);

        for (int i = 0; i < size\_cnt; i++)

        {

            for (int j = 0; j < table[i].size(); j++)

            {

                if (table[i][j].nameSP == element)

                {

                    cout << endl << "Элемент успешно найден, его номер: " << position << endl;

                    return position;

                }

                position++;

            }

        }

        cout << endl << "Не удалось найти элемент" << endl;

        return -1;

    }

    void add\_element()

    {

        Human temp\_human;

        cout << "Введите ФИО: ";

        cin.seekg(cin.eof());

        getline(cin, temp\_human.nameSP);

        cout << "Введите дату рождения: ";

        cin.seekg(cin.eof());

        getline(cin, temp\_human.birthday);

        cout << "Введите номер телефона: ";

        cin.seekg(cin.eof());

        getline(cin, temp\_human.phoneNum);

        int index = getHash(temp\_human.nameSP);

        table[index].push\_back(temp\_human);

        size\_cnt++;

    }

    void remove\_element(string element)

    {

        int index = getHash(element);

        for (int j = 0; j < table[index].size(); j++)

        {

            if (table[index][j].nameSP == element)

            {

                table[index].erase(table[index].begin() + j);

                size\_cnt--;

                cout << endl << "Элемент успешно удален" << endl;

                return;

            }

        }

        cout << endl << "Не удалось найти элемент" << endl;

    }

};

int main()

{

    srand(time(0));

    cout << "Кол-во человек: ";

    cin >> hashTable\_size;

    Hash\_Table people;

    people.table.resize(hashTable\_size);

    for (int i = 0; i < hashTable\_size; i++) // заполнение

    {

        Human temp\_human;

        temp\_human.nameSP = generate\_fullName();

        temp\_human.birthday = generate\_date();

        temp\_human.phoneNum = generate\_PhoneNum();

        int index = getHash(temp\_human.nameSP);

        if (people.table[index].size() != 0)

        {

            people.collision\_cnt++;

        }

        people.table[index].push\_back(temp\_human);

        people.size\_cnt++;

    }

    cout << "Количество колллизий: " << people.collision\_cnt << endl;

    people.print();

    cout << "Введите ключ для удаления [ФИО]: ";

    string temp\_string;

    cin.seekg(cin.eof());

    getline(cin, temp\_string);

    people.remove\_element(temp\_string);

    people.print();

    cout << "\tДобавление элемента:" << endl;

    people.add\_element();

    people.print();

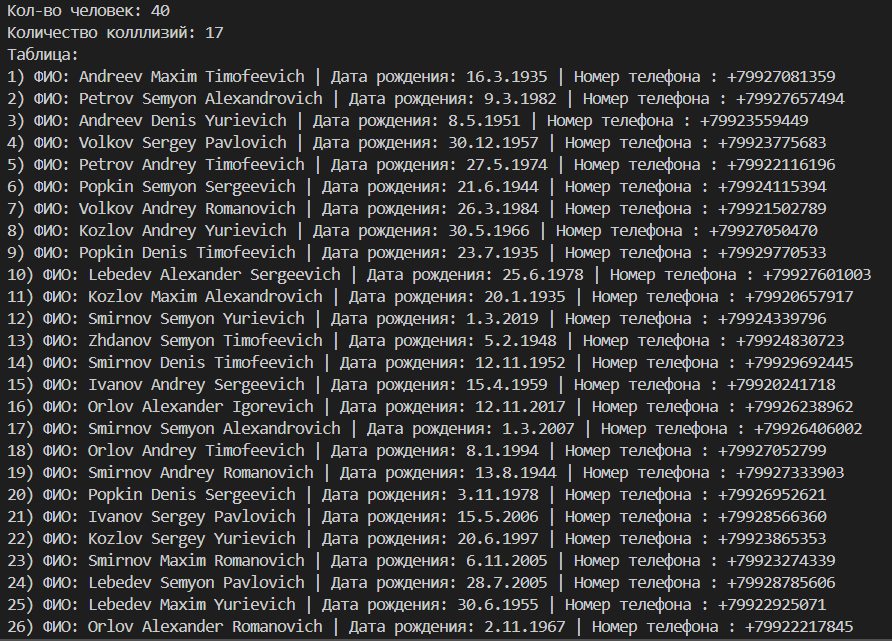
    cout << "Введите ключ для поиска [ФИО]: ";

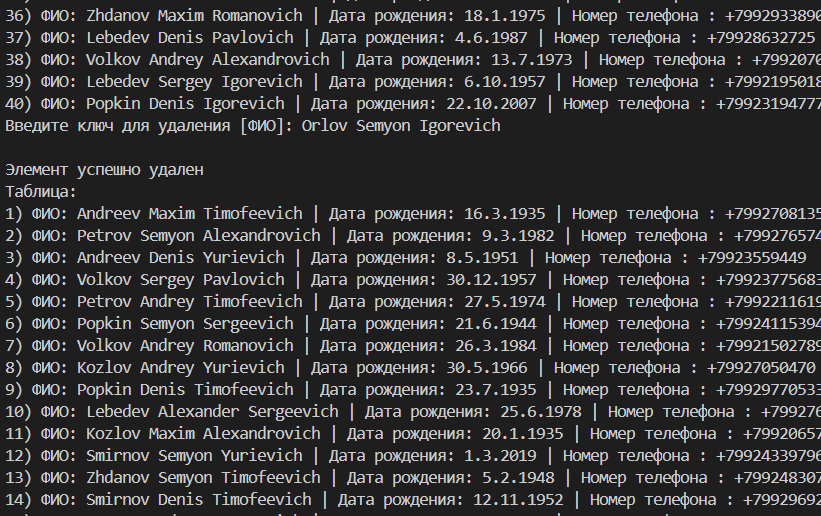
    cin.seekg(cin.eof());

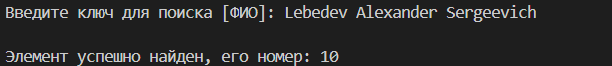
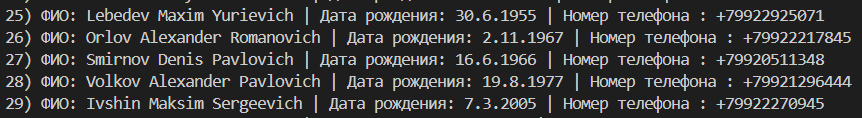
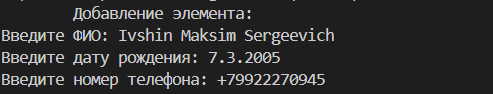
    getline(cin, temp\_string);

    people.find\_element(temp\_string);

}

**Пример работы программы:**

****

****

**GitHub:** *https://github.com/geroineee/PNRPU*