Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

Лабораторная работа №11

Поиск данных с помощью хэш-таблиц

Выполнил

Студент группы РИС-22-2б

Вековшинин Д. А.

Проверила

Доцент кафедры ИТАС

Полякова О. А.

Пермь 2023

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.

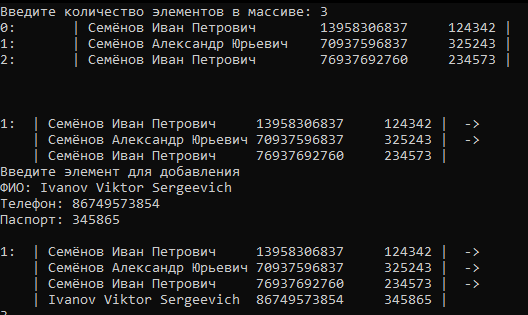
3. Предусмотреть возможность добавления и удаления элементов из массива (файла).

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

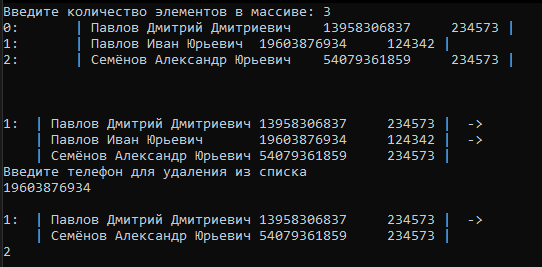
5. Подсчитать количество коллизий при размере хэш-таблицы 40, 75 и 90 элементов.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 | ФИО, №паспорта, №телефона | №телефона | H(k)= [M (kAmod1)], 0<A<1, mod1 – получение дробной части, [] – получение целой части | Метод открытой адресации |

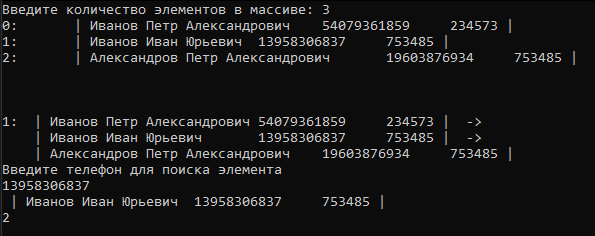
Тест функции добавления элемента

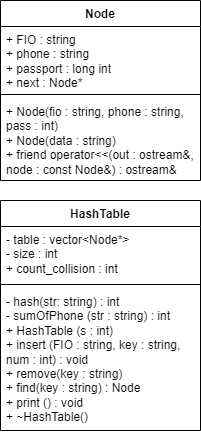


Тест функции для удаления элемента



Тест функции для поиска элемента по ключу



****

Код программы

#include <iostream>

#include <vector>

#include <string>

#include <fstream>

#include <stack>

using namespace std;

string names[5] = { "Иван", "Александр", "Петр", "Дмитрий", "Денис" };

string patronymic[5] = { "Иванович", "Александрович", "Петрович", "Дмитриевич", "Юрьевич" };

string surnames[5] = { "Иванов", "Александров", "Петров", "Семёнов", "Павлов" };

int passport[5] = { 753485, 234573, 124383, 325243, 124342 };

string phone[5] = { "76937692760", "13958306837", "19603876934", "70937596837", "54079361859" };

struct Node {

string FIO;

string phone;

long int passport;

Node\* next;

Node(string fio, string phone, int pass) : FIO(fio), phone(phone), passport(pass), next(nullptr) {}

Node(string data) : phone(data), FIO("NULL"), passport(0), next(nullptr) {}

friend ostream& operator<<(ostream& out, const Node& node)

{

if (node.phone == "not found") { out << "not found"; }

else if (node.phone == "delete") { out << "delete"; }

else

out << " | " << node.FIO << "\t" << node.phone << "\t" << node.passport << " | ";

return out;

}

};

class HashTable {

private:

vector<Node\*> table;

int size;

int hash(string str)

{

double a = (10/2486) \* sumOfPhone(str);

double c = size \* (a - int(a));

return int(c);

}

int sumOfPhone(string str)

{

int sum = 0;

for (int i = 0; i < 10; i++)

{

sum += stoi(str.substr(i, 2));

}

return sum;

}

public:

int count\_collision = 0;

HashTable(int s) : size(s)

{

table.resize(size, nullptr);

}

void insert(string FIO, string key, int num) {

int index = hash(key);

Node\* newNode = new Node(FIO, key, num);

if (table[index] == nullptr) {

table[index] = newNode;

}

else {

count\_collision++;

Node\* current = table[index];

while (current->next != nullptr) {

/\*if (current->key == key) {

current->value = value;

return;

}\*/

current = current->next;

}

/\*if (current->key == key) {

current->value = value;

return;

}\*/

current->next = newNode;

}

}

void remove(string key) {

int index = hash(key);

if (table[index] == nullptr) {

return;

}

else if (table[index]->phone == key) {

Node\* temp = table[index];

table[index] = temp->next;

delete temp;

}

else {

Node\* current = table[index];

while (current->next != nullptr && current->next->phone != key) {

current = current->next;

}

if (current->next != nullptr && current->next->phone == key) {

Node\* temp = current->next;

current->next = temp->next;

delete temp;

}

}

}

Node find(string key) {

int index = hash(key);

Node\* current = table[index];

while (current != nullptr) {

if (current->phone == key) {

return \*current;

}

current = current->next;

}

return Node("not found");

}

void print() {

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

Node\* current = table[i];

if (current != nullptr) {

cout << endl << i + 1 << ": " << \*current;

current = current->next;

}

while (current != nullptr) {

cout << " ->\n " << \*current;

current = current->next;

}

}

}

~HashTable() {

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

Node\* current = table[i];

while (current != nullptr) {

Node\* temp = current;

current = current->next;

delete temp;

}

}

}

};

int randomNum()

{

return rand() % 5;

}

string randomName()

{

return (surnames[randomNum()] + " " + names[randomNum()] + " " + patronymic[randomNum()]);

}

long int randomPassport()

{

return passport[randomNum()];

}

string randomPhone()

{

return phone[randomNum()];

}

Node\* createRandomPerson()

{

Node\* temp = new Node("not find");

temp->FIO = randomName();

temp->passport = randomPassport();

temp->phone = randomPhone();

return temp;

}

void fillArray(vector<Node\*>& array, const int size)

{

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

array[i] = createRandomPerson();

}

int main()

{

setlocale(LC\_ALL, "ru");

int size;

cout << "Введите количество элементов в массиве: ";

cin >> size;

vector<Node\*> array;

array.resize(size);

HashTable Table(size);

fillArray(array, size);

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

{

cout << i << ":\t" << \*array[i] << endl;

string FIO = array[i]->FIO;

string data = array[i]->phone;

long int num = array[i]->passport;

Table.insert(FIO, data, num);

}

cout << endl << endl;

Table.print();

cout << endl << Table.count\_collision << endl;

return 0;

}