Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

**Пермский национальный исследовательский политехнический университет**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: «Хэш – таблицы»

Семестр 2

*Работу выполнил: Учащийся группы ИВТ - 22-2б: Цыбуцынин Фёдор Александрович*

*Работу проверил: доцент кафедры ИТАС:*

*Полякова Ольга Андреевна*

Г. 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 возможность добавления и удаления элементов из массива.
3. Выполнить поиск элемента в массиве по ключу в соответствии с вариантом. Для поиска использовать хэш-таблицу.
4. Подсчитать количество коллизий при размере хэш-таблицы 40, 75 и 90 элементов.

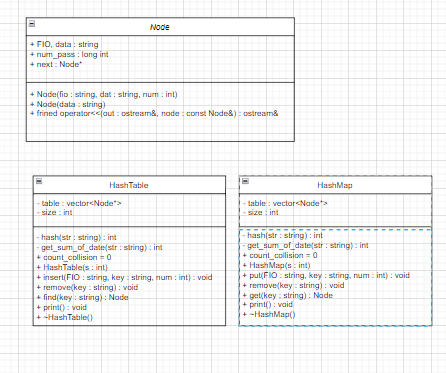
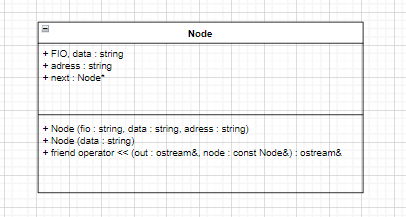
ВАРИАНТ 13:



**Алгоритм решения задачи**

1. Создается структура Node, которая содержит в себе информацию о человеке (ФИО, дата рождения, адрес).
2. Создаются функции, которые заполняют хэш-таблицу методами открытой адресации и методом цепочек.
3. Функция get\_random\_name составляет ФИО человека из набора случайным образом.
4. Функция get\_random\_address добавляет в информацию о человеке адрес проживания случайным образом.
5. Функция get\_random\_date добавляет в информацию о человеке дату его рождения случайным образом.
6. Функция create\_random\_human «собирает» все данные о человеке.
7. Функция fill\_array\_random\_human создает массив данных о людях.
8. Выводится готовая Хэш-таблица и количество коллизий.

**UML диаграмма**

****

**Рис. 1 – UML диаграмма**

**Код программы**

#include <iostream>

#include <vector>

#include <string>

#include <fstream>

#include <stack>

using namespace std;

string names[5] = { "Михаил", "Павел", "Юрий","Иван", "Роман" };

string patronymic[5] = { "Алексеевиич", "Николаевич", "Евгеньевич","Александрович", "Сергеевич" };

string surnames[5] = { "Петров", "Юсупов", "Сидоров", "Моссунов", "Иванов" };

string address[5] = { "ул.Мира 115", "ул.Ленина 56", "ул.Попова 17", "ул.Куйбышева 82", "ул.Хохрякова 27" };

string dates[5] = { "15.09.1969", "27.01.1996", "23.10.1972", "13.04.1993", "20.07.1985" };

struct Node {

string FIO;

string data;

string address;

Node\* next;

Node(string fio, string dat, string add) : FIO(fio), data(dat), address(add), next(nullptr) {}

Node(string data) : data(data), FIO("NULL"), address("NULL"), next(nullptr) {}

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

{

if (node.data == "not find") { out << "not find"; }

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

else

out << " | " << node.FIO << "\t" << node.data << "\t" << node.address << " | ";

return out;

}

};

class HashTable {

private:

vector <Node\*> table;

int size;

int hash(string str)

{

double a = ((sqrt(5) - 1) / 2) \* get\_sum\_of\_date(str); // a = 0.618 \* 2015 == 1245.27

double c = size \* (a - int(a)); // с = 10 \* (1245.27 - 1245) == 2.7

return int(c); // с == 2

}

int get\_sum\_of\_date(string str)

{

return stoi(str.substr(0, 2)) + stoi(str.substr(3, 2)) + stoi(str.substr(6, 4));

}

public:

int count\_collision = 0;

HashTable(int s) : size(s)

{

table.resize(size, nullptr);

}

void insert(string FIO, string key, string add) {

int index = hash(key);

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

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]->data == key) {

Node\* temp = table[index];

table[index] = temp->next;

delete temp;

}

else {

Node\* current = table[index];

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

current = current->next;

}

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

return \*current;

}

current = current->next;

}

return Node("not find");

}

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;

}

}

}

};

class HashMap {

private:

vector<Node\*> table;

int size;

int hash(string str)

{

double a = ((sqrt(5) - 1) / 2) \* get\_sum\_of\_date(str); // a = 0.618 \* 2015 == 1245.27

double c = size \* (a - int(a)); // с = 10 \* (1245.27 - 1245) == 2.7

return int(c); // с == 2

}

int get\_sum\_of\_date(string str)

{

return stoi(str.substr(0, 2)) + stoi(str.substr(3, 2)) + stoi(str.substr(6, 4));

}

public:

int count\_collision = 0;

HashMap(int s) : size(s)

{

table.resize(size, nullptr);

}

~HashMap() {

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

{

if (table[i] != nullptr)

delete table[i];

}

}

void put(string FIO, string key, string add) {

int index = hash(key);

int end = index;

if (table[index] == nullptr)

{

table[index] = new Node(FIO, key, add);

return;

}

else

{

count\_collision++;

++index;

while (index < size)

{

if (table[index] == nullptr)

{

table[index] = new Node(FIO, key, add);

return;

}

++index;

}

if (index == size)

{

index = 0;

while (index < end)

{

if (table[index] == nullptr)

{

table[index] = new Node(FIO, key, add);

return;

}

++index;

}

}

}

}

Node get(string key) {

int index = hash(key);

while (table[index] != nullptr && table[index]->data != key) {

index = (index + 1) % size;

}

if (table[index] == nullptr)

return Node("not find");

else

return \*table[index];

}

void remove(string key) {

int index = hash(key);

while (table[index] != nullptr) {

if (table[index]->data == key)

break;

index = (index + 1) % size;

}

if (table[index] == nullptr)

return;

else {

delete table[index];

table[index] = nullptr;

}

}

void print() {

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

if (table[i] != nullptr)

cout << i + 1 << ": " << \*table[i] << endl;

}

}

};

int get\_random\_addber()

{

return rand() % 5;

}

string get\_random\_name()

{

return (surnames[get\_random\_addber()] + " " + names[get\_random\_addber()] + " " + patronymic[get\_random\_addber()]);

}

string get\_random\_address()

{

return address[get\_random\_addber()];

}

string get\_random\_date()

{

return dates[get\_random\_addber()];

}

Node\* create\_random\_human()

{

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

temp->FIO = get\_random\_name();

temp->address = get\_random\_address();

temp->data = get\_random\_date();

return temp;

}

void fill\_array\_random\_humans(vector<Node\*>& array, const int size)

{

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

array[i] = create\_random\_human();

}

int main()

{

setlocale(0, ".1251");

int size;

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

cin >> size;

cout << endl;

vector<Node\*> array;

array.resize(size);

HashMap Table1(size);

HashTable Table2(size);

fill\_array\_random\_humans(array, size);

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

{

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

string FIO = array[i]->FIO;

string data = array[i]->data;

string add = array[i]->address;

Table2.insert(FIO, data, add);

Table1.put(FIO, data, add);

}

cout << endl << endl;

Table2.print();

cout << endl << endl;

Table1.print();

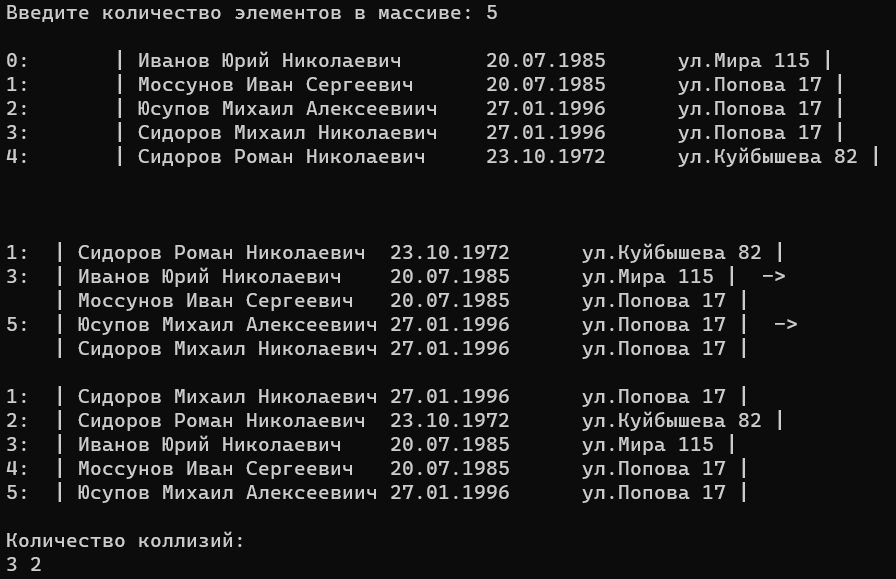
cout << endl << "Количество коллизий: ";

cout << endl << Table1.count\_collision << " " << Table2.count\_collision << endl;

return 0;

}

**Результаты работы программы**

****

**Рис. 2 – Вывод консоли программы**