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-23-3б

Коротаев Александр Дмитриевич

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 саму 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**

Hash.h

#pragma once

double A = 0.143592753589793;

int getHash(int k)

{

return static\_cast<int>(M \* (k \* A - static\_cast<int>(k \* A)));

}

int getHash(string text)

{

int result = 0;

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

{

result += static\_cast<int>(pow(text[i], 2) \* (2 / sqrt(M)) + abs(text[i] \* (1 / sqrt(2))));

}

return getHash(abs(result));

}

Generators.h

#pragma once

string correct\_date(int n)

{

string date = to\_string(n);

if (date.size() < 2)

{

date = '0' + date;

}

return date;

}

string generate\_date()

{

return correct\_date(rand() % 30 + 1) + "." + correct\_date(rand() % 12 + 1) + "." + to\_string(2024 - (rand() % 90));

}

string generate\_pasNum()

{

return to\_string(rand()%900000\*10 + 100000 + rand()%10);

}

string generate\_fullName()

{

string surnames[] = { "Иванов", "Карпов", "Козлов", "Орлов", "Жданов", "Волков", "Попкин", "Петров", "Андреев", "Лебедев", "Макаров" };

string names[] = { "Александр", "Сергей", "Андрей", "Семен", "Денис", "Максим" };

string patronymics[] = { "Александрович", "Игоревич", "Павлович", "Юрьевич", "Сергеевич", "Романович", "Тимофеевич" };

return surnames[rand() % 11] + " " + names[rand() % 6] + " " + patronymics[rand() % 7];

}

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

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

using namespace std;

int hashTable\_size;

extern int& M = hashTable\_size;

#include "Hash.h"

#include "Generators.h"

struct Human

{

string nameSP = "-";

string birthday = "-";

string pas\_num = "-";

};

struct Hash\_Table

{

vector<Human> table;

int collision\_cnt = 0;

void print()

{

cout << "Таблица: \n";

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 паспорта: " << table[i].pas\_num << endl;

position++;

}

if (position == 1)

{

cout << "Таблица пуста\n";

}

}

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

{

collision\_cnt++;

int cnt = 0;

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

{

index++;

if (index == M)

{

index = 0;

}

cnt++;

}

if (cnt == M)

cout << "\nОшибка устранения коллизии\n";

else

table[index] = new\_element;

}

int find\_element(string& element)

{

int index = getHash(element);

int cnt = 0;

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

{

index++;

if (index == M)

{

index = 0;

}

cnt++;

}

if (cnt >= M)

{

return -1;

}

else

{

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 паспорта: ";

cin.seekg(cin.eof());

getline(cin, temp\_human.pas\_num);

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

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

{

collision\_solve(index, temp\_human);

}

else

{

table[index] = temp\_human;

}

}

void remove\_element(string& element)

{

int index = find\_element(element);

if (index == -1)

{

cout << "Не удалось найти элемент\n";

}

else

{

table[index].nameSP = "-";

table[index].pas\_num = "-";

table[index].birthday = "-";

cout << "\nЭлемент успешно удален\n";

}

}

void remove\_element(int& index)

{

table[index].nameSP = "-";

table[index].pas\_num = "-";

table[index].birthday = "-";

}

void delete\_table()

{

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

{

remove\_element(i);

}

cout << "\nУдаление таблицы...\n\n";

}

void file\_print(string file)

{

ofstream out (file);

out << "Таблица: \n";

int position = 1;

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

{

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

continue;

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

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

out << " | Номер паспорта: " << table[i].pas\_num << endl;

position++;

}

out.close();

}

void recovery\_table(string file)

{

cout << "\nВосстановление таблицы...\n\n";

ifstream in(file);

Human temp\_human;

string str;

getline(in, str);

while (getline(in, str))

{

int index\_1 = 0, index\_2;

for (index\_1; index\_1 < str.length(); index\_1++)

{

if (str[index\_1] == ':')

{

index\_2 = index\_1;

}

if (str[index\_1] == '|')

{

break;

}

}

temp\_human.nameSP = str.substr(index\_2 + 2, index\_1 - 3 - index\_2);

temp\_human.birthday = str.substr(index\_1 + 17, 10);

temp\_human.pas\_num = str.substr(index\_1 + 46, 6);

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

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

{

collision\_solve(index, temp\_human);

}

else

{

table[index] = temp\_human;

}

}

in.close();

file\_print("F2.txt");

}

};

int main()

{

system("chcp 1251>null");

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++)

{

Human temp\_human;

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

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

temp\_human.pas\_num = generate\_pasNum();

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 [Дата рождения]: ";

string temp\_string;

cin.seekg(cin.eof());

getline(cin, temp\_string);

people.remove\_element(temp\_string);

people.print();

cout << "\tДобавление элемента:\n";

people.add\_element();

people.print();

cout << "Введите ключ для поиска [Дата рождения]: ";

cin.seekg(cin.eof());

getline(cin, temp\_string);

int index = people.find\_element(temp\_string);

if (index == -1)

{

cout << "Не удалось найти элемент\n";

}

else

{

cout << "\nЭлемент найден, его номер: " << index+1 << endl;

}

people.file\_print("F1.txt");

people.delete\_table();

people.print();

people.recovery\_table("F1.txt");

people.print();

}

**Код на С++ для метода цепочек:**

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

using namespace std;

int hashTable\_size;

extern int& M = hashTable\_size;

#include "Hash.h"

#include "Generators.h"

struct Human

{

string nameSP = "-";

string birthday = "-";

string pas\_num = "-";

};

struct Hash\_Table

{

vector<vector<Human>> table;

int collision\_cnt = 0;

bool is\_empty = false;

void print()

{

cout << "Таблица: " << endl;

int position = 1;

if (!is\_empty)

{

for (int i = 0; i < table.size(); 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 паспорта: " << table[i][j].pas\_num << endl;

position++;

}

}

}

else

{

cout << "Таблица пуста\n";

}

}

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

{

collision\_cnt++;

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

}

int find\_element(string element)

{

int position = 0;

int index = getHash(element);

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

{

position += table[i].size();

}

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

{

if (table[index][i].birthday == element)

{

return position;

}

position++;

}

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 паспорта: ";

cin.seekg(cin.eof());

getline(cin, temp\_human.pas\_num);

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

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

}

void remove\_element(string element)

{

int index = getHash(element);

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

{

if (table[index][i].birthday == element)

{

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

cout << "\nЭлемент успешно удален\n";

return;

}

}

cout << "\nНе удалось найти элемент\n";

}

void delete\_table()

{

cout << "\nУдаление таблицы...\n\n";

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

{

table[i].clear();

}

is\_empty = true;

}

void file\_print(string file)

{

ofstream out(file);

out << "Таблица: " << endl;

int position = 1;

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

{

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

{

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

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

out << " | Номер паспорта: " << table[i][j].pas\_num << endl;

position++;

}

}

out.close();

}

void recovery\_table(string file)

{

cout << "\nВосстановление таблицы...\n\n";

ifstream in(file);

Human temp\_human;

string str;

getline(in, str);

while (getline(in, str))

{

int index\_1 = 0, index\_2;

for (index\_1; index\_1 < str.length(); index\_1++)

{

if (str[index\_1] == ':')

{

index\_2 = index\_1;

}

if (str[index\_1] == '|')

{

break;

}

}

temp\_human.nameSP = str.substr(index\_2 + 2, index\_1 - 3 - index\_2);

temp\_human.birthday = str.substr(index\_1 + 17, 10);

temp\_human.pas\_num = str.substr(index\_1 + 46, 6);

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

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

}

is\_empty = false;

in.close();

file\_print("F2.txt");

}

};

int main()

{

system("chcp 1251>null");

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++)

{

Human temp\_human;

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

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

temp\_human.pas\_num = generate\_pasNum();

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

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

{

people.collision\_cnt++;

}

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

}

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

people.print();

cout << "Введите ключ для удаления [Дата рождения]: ";

string temp\_string;

cin.seekg(cin.eof());

getline(cin, temp\_string);

people.remove\_element(temp\_string);

cout << "\tДобавление элемента:\n";

people.add\_element();

cout << endl;

people.print();

cout << "Введите ключ для поиска [Дата рождения]: ";

cin.seekg(cin.eof());

getline(cin, temp\_string);

int index = people.find\_element(temp\_string);

if (index == -1)

{

cout << "Не удалось найти элемент\n";

}

else

{

cout << "\nЭлемент найден, его номер: " << index + 1 << endl;

}

people.file\_print("F1.txt");

people.delete\_table();

people.print();

people.recovery\_table("F1.txt");

people.print();

}

**Вывод**

По итогам данной работы я узнал что такое хэш таблицы и научился с ними работать и освоил методы разрешения коллизий в хэш таблицах: метод открытой адресации и метод цепочек. Благодаря этому я научился эффективно манипулировать данными.

**GitHub**

[*https://github.com/Korovay4ik/Laboratory-works*](https://github.com/Korovay4ik/Laboratory-works)

**Результаты работы программы**

