Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное

учреждение

высшего образования

«**Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные

системы»

**ОТЧЁТ**

**по лабораторной работе №17**

Дисциплина: «Основы теории алгоритмов и структуры данных»

Тема: Методы быстрого поиска. Хеширование

Вариант 17

Выполнил:

Студент группы ИВТ-20-2б

Сафронов Владислав Владиславович

Проверила:

Доцент кафедры ИТАС

Полякова О.А.

**Пермь, 2021**

**Цель задачи**

1. Изучить построение функции хеширования и алгоритмов хеширования данных
2. Научиться разрабатывать алгоритмы открытого и закрытого хеширования при решении задач на языке C++.

**Постановка задачи**

1. Создать динамический массив из записей (ФИО, дата рождения, номер телефона), содержащий не менее 100 элементов. Для заполнения элементов массива использовать ДСЧ.
2. Выполнить поиск элемента в массиве по ключу в соответствии с вариантом (ФИО). Для поиска использовать хеш-таблицу.

**Анализ задачи**

**Какие типы данных будут использованы:**

* Целочисленный тип данных integer используется для хранения и ввода размера массива, индексов элементов, параметра в циклах.
* Строковый тип данных string для ввода и хранения ФИО и телефонных номеров.

**Какие с этими данными выполняются действия:**

* integer – ввод с клавиатуры, инкремент, присваивание, сравнение;
* String – ввод с клавиатуры, сравнение, инкремент.

**В каком виде эти данные будут представлены:**

1. Структура Date для хранения даты рождения. Имеет 3 целочисленных поля: day, mon и yr для хранения дня, месяца и года рождения соответсвенно:

struct Date

{

int day, mon, yr;

};

1. Структура Human. Имеет 3 поля: строковое поле name для хранения ФИО, экземпляр структуры Date для хранения даты рождения и строковое поле phone для хранения номера телефона:

struct Human

{

string name;

Date birth;

string phone;

* 1. Метод структуры Print для вывода элемента структуры:

void Print()

{

cout << "ФИО: " << name;

cout << "\nДата рождения: " << birth.day << "/" << birth.mon << "/" << birth.yr;

cout << "\nНомер телефона: " << phone;

cout << "\n--------------------------------------------------------\n";

}

* 1. Метод структуры GeteratePhone для генерации номера телефона. Использует датчик случайных чисел. Возвращает строку со сгенерированным номером телефона:

string GeneratePhone()

{

string res = "+79";

for (int i = 0; i < 9; i++)

{

res += to\_string(rand() % 10);

}

return res;

}

* 1. Метод Generate для заполнения данных элемента структуры с помощью датчика случайных чисел:

void Generate()

{

string names[] = { "Марк", "Егор", "Руслан", "Андрей", "Дмитрий", "Павел", "Юрий", "Сергей", "Никита", "Кирилл" };

string snames[] = { "Терехов", "Латышев", "Адрианов", "Титов", "Логинов", "Тихонов", "Беликов", "Поздняков", "Блинов", "Исаков" };

string fnames[] = { "Маркович", "Егорович", "Русланович", "Андреевич", "Дмитриевич", "Павлович", "Юрьевич", "Сергеевич", "Никитович", "Кириллович" };

name = snames[rand() % 10] + " " + names[rand() % 10] + " " + fnames[rand() % 10];

birth.mon = rand() % 12 + 1;

if (birth.mon == 2) birth.day = rand() % 28 + 1;

else birth.day = rand() % 30 + 1;

birth.yr = 1900 + (rand() % 120 + 1);

phone = GeneratePhone();

}

1. Структура HashTable для создания элемента хеш-таблицы. Имеет 2 поля: целочисленное поле key и массив экземпляров структуры Human с одинаковым хешем:

struct HashTable

{

int key;

vector<Human> list;

* 1. Метод HashFunc для создания ключа по длине строки. На вход функция принимает строку с ФИО элемента и количество элементов. Ключ вычисляется делением суммы численных значений всех символов строки на количество элементов:

void HashFunc(string name, int size)

{

int sum = 0;

for (int i = 0; i < name.length(); i++)

{

sum += abs(int(name[i]));

}

key = sum % size;

}

* 1. Метод Add для добавления элемента в массив:

void Add(Human a) {

list.push\_back(a);

}

1. Функция CreateArr для создания и заполнения данными массива элементов структуры Human. В качестве входного параметра принимает переменную с размером массива (по умолчанию – 100). Возвращает сгенерированный список:

vector<Human> CreateArr(int size = 100)

{

vector<Human> lst;

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

{

Human elem;

elem.Generate();

lst.push\_back(elem);

}

return lst;

}

1. Функция Sort для сортировки хеш-таблицы по ключу:

void Sort(vector<HashTable>& lst)

{

for (int st = lst.size() / 2; st > 0; st /= 2)

{

for (int i = st; i < lst.size(); i++)

{

for (int j = i; j >= st; j -= st)

{

if (lst[j].key < lst[j - st].key) swap(lst[j].key, lst[j - st].key);

}

}

}

}

1. Функция PrintArr для вывода массива в консоль:

void PrintArr(vector<Human> lst)

{

if (lst.empty())

{

cout << "Список пуст.\n";

return;

}

cout << "\n--------------------------------------------------------\n";

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

{

cout << "Номер: " << i + 1 << endl;

lst[i].Print();

}

}

1. Функция PrintHashTable для вывода хеш-таблицы в консоль:

void PrintHashTable(vector<HashTable> hash\_lst)

{

for (int i = 0; i < hash\_lst.size(); i++)

{

cout<<"===========================================================\n";

cout << "Хеш " << i + 1 << ":\t" << hash\_lst[i].key << endl;

cout << "Элементы с этим хешем: \t" << hash\_lst[i].list.size() << endl;

for (int j = 0; j < hash\_lst[i].list.size(); j++)

{

cout << "Номер " << j + 1 << endl;

cout << "ФИО:\t" << hash\_lst[i].list[j].name << endl;

cout << "Дата рождения:\t" << hash\_lst[i].list[j].birth.day << "/" << hash\_lst[i].list[j].birth.mon << "/" << hash\_lst[i].list[j].birth.yr << endl;

cout << "Номер телефона:\t" << hash\_lst[i].list[j].phone << endl;

cout << "Hash:\t" << hash\_lst[i].key << endl;

cout << endl;

}

}

}

1. Функция FillHashTable для заполнения хеш-таблицы. В качестве входных параметров функция принимает исходный массив и ссылку на массив хеш-таблицу. Сначала в цикле каждому элементу исходного массива присваивается ключ и добавляется в хеш-таблицу. Затем хеш-таблица сортируется и в следующем цикле происходит поиск и объединение элементов хеш-таблицы с одинаковыми ключами:

void FillHashTable(vector<Human> lst, vector<HashTable>& hash\_lst)

{

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

{

HashTable tmp;

tmp.HashFunc(lst[i].name, lst.size());

tmp.Add(lst[i]);

hash\_lst.push\_back(tmp);

}

Sort(hash\_lst);

int len = hash\_lst.size();

for (int i = 0; i < len - 1;)

{

if (hash\_lst[i].key == hash\_lst[i + 1].key)

{

hash\_lst[i].list.push\_back(hash\_lst[i + 1].list[0]);

hash\_lst.erase(hash\_lst.begin() + i + 1);

len--;

}

else i++;

}

}

1. Функция FindByName для поиска элемента в хеш-таблице по ФИО. В качестве входных параметров принимается хеш-таблица, строка с ФИО, и размер массива. Для введенного ФИО с помощью хеш-функции создается ключ и по этому ключу идет поиск в хеш-таблице. Если в таблице есть элементы с данным хешем, то веденный ФИО сравнивается с каждым элементом с таким же ключом. В случае успешного нахождения элемента, он будет выведен в консоль:

void FindByName(vector<HashTable> hash\_lst, string key, int size)

{

int sum = 0;

for (int i = 0; i < key.length(); i++)

{

sum += abs(int(key[i]));

}

int key\_hash = sum % size;

for (int i = 0; i < hash\_lst.size(); i++)

{

if (hash\_lst[i].key == key\_hash)

{

cout << "Элемент найден!\n";

for (int j = 0; j < hash\_lst[i].list.size(); j++)

{

if (key == hash\_lst[i].list[j].name)

{

cout << "ФИО:\t" << hash\_lst[i].list[j].name << endl;

cout << "Дата рождения:\t" << hash\_lst[i].list[j].birth.day << "/" << hash\_lst[i].list[j].birth.mon << "/" << hash\_lst[i].list[j].birth.yr << endl;

cout << "Номер телефона:\t" << hash\_lst[i].list[j].phone;

cout << "Hash:\t" << hash\_lst[i].key << endl;

}

}

cout << "===========================================================\n";

return;

}

}

cout << "Элементов не найдено.\n";

}

1. Функция FindByHash для поиска элементов в таблице по хешу. В качестве входных параметров функция принимает хеш-таблицу и ключ. В цикле идет поиск элемента хеш-таблицы с введенным ключом. В случае успеха выведятся элементы таблицы с искомым ключом:

void FindByHash(vector<HashTable> hash\_lst, int key)

{

for (int i = 0; hash\_lst.size(); i++)

{

if (key == hash\_lst[i].key)

{

cout << "Элемент(ы) с таким хешем найден(ы):\n";

PrintArr(hash\_lst[i].list);

cout << "===========================================================\n";

return;

}

}

cout << "Элементов не найдено.\n";

}

**Какими операторами будет организован ввод и вывод:**

* Ввод данных осуществляется через консоль с помощью функции cin и getline();
* Вывод данных в консоль осуществляется с помощью функции cout.

**Описание функции main():**

Подключение русского языка:

setlocale(LC\_ALL, "Russian");

Функция srand для работы с генератором случайных чисел:

srand(time(NULL));

Резервирование памяти под переменные, создание и заполнение данными массива элементов структуры Human, создание и заполнение хеш-таблицы:

vector<Human> list = CreateArr();

vector<HashTable> hash\_list;

PrintArr(list);

FillHashTable(list, hash\_list);

Вывод хеш-таблицы:

cout << "Таблица хешей:\n";

PrintHashTable(hash\_list);

Запрос на выбор метода поиска:

cout << "Методы поиска:\n";

cout << "| 1. Поиск по имени.\n";

cout << "| 2. Поиск по хешу.\n";

cout << "Выберите метод поиска (1-2): ";

cin >> n;

cin.get();

Осуществление поиска в хеш-таблице в зависимости от выбора пользователя, либо завершение работы программы в случае некорректного ввода:

switch (n)

{

case 1:

{

string key;

cout << "Введите ФИО для поиска: ";

getline(cin, key);

while (key == "" || key[0] == ' ')

{

cout << "\nФИО введено некорректно.\n";

cout << "Введите ФИО для поиска: ";

getline(cin, key);

}

FindByName(hash\_list, key, list.size());

break;

}

case 2:

{

int key;

cout << "Введите хеш для поиска: ";

cin >> key;

FindByHash(hash\_list, key);

break;

}

default:

{

cout << "Ввод некорректен. Завершение работы...\n";

}

}

}

**Блок-схема программы**

**Код программы**

#include <iostream>

#include <string>

#include <vector>

#include <list>

#include <ctime>

#include <cmath>

using namespace std;

struct Date

{

int day, mon, yr;

};

struct Human

{

string name;

Date birth;

string phone;

void Print()

{

cout << "ФИО: " << name;

cout << "\nДата рождения: " << birth.day << "/" << birth.mon << "/" << birth.yr;

cout << "\nНомер телефона: " << phone;

cout << "\n--------------------------------------------------------\n";

}

string GeneratePhone()

{

string res = "+79";

for (int i = 0; i < 9; i++)

{

res += to\_string(rand() % 10);

}

return res;

}

void Generate()

{

string names[] = { "Марк", "Егор", "Руслан", "Андрей", "Дмитрий", "Павел", "Юрий", "Сергей", "Никита", "Кирилл" };

string snames[] = { "Терехов", "Латышев", "Адрианов", "Титов", "Логинов", "Тихонов", "Беликов", "Поздняков", "Блинов", "Исаков" };

string fnames[] = { "Маркович", "Егорович", "Русланович", "Андреевич", "Дмитриевич", "Павлович", "Юрьевич", "Сергеевич", "Никитович", "Кириллович" };

name = snames[rand() % 10] + " " + names[rand() % 10] + " " + fnames[rand() % 10];

birth.mon = rand() % 12 + 1;

if (birth.mon == 2) birth.day = rand() % 28 + 1;

else birth.day = rand() % 30 + 1;

birth.yr = 1900 + (rand() % 120 + 1);

phone = GeneratePhone();

}

};

struct HashTable

{

int key;

vector<Human> list;

void HashFunc(string name, int size)

{

int sum = 0;

for (int i = 0; i < name.length(); i++)

{

sum += abs(int(name[i]));

}

key = sum % size;

}

void Add(Human a)

{

list.push\_back(a);

}

};

vector<Human> CreateArr(int size = 100)

{

vector<Human> lst;

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

{

Human elem;

elem.Generate();

lst.push\_back(elem);

}

return lst;

}

void Sort(vector<HashTable>& lst)

{

for (int st = lst.size() / 2; st > 0; st /= 2)

{

for (int i = st; i < lst.size(); i++)

{

for (int j = i; j >= st; j -= st)

{

if (lst[j].key < lst[j - st].key) swap(lst[j].key, lst[j - st].key);

}

}

}

}

void PrintArr(vector<Human> lst)

{

if (lst.empty())

{

cout << "Список пуст.\n";

return;

}

cout << "\n--------------------------------------------------------\n";

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

{

cout << "Номер: " << i + 1 << endl;

lst[i].Print();

}

}

void PrintHashTable(vector<HashTable> hash\_lst)

{

for (int i = 0; i < hash\_lst.size(); i++)

{

cout << "===========================================================\n";

cout << "Хеш " << i + 1 << ":\t" << hash\_lst[i].key << endl;

cout << "Элементы с этим хешем: \t" << hash\_lst[i].list.size() << endl;

for (int j = 0; j < hash\_lst[i].list.size(); j++)

{

cout << "Номер " << j + 1 << endl;

cout << "ФИО:\t" << hash\_lst[i].list[j].name << endl;

cout << "Дата рождения:\t" << hash\_lst[i].list[j].birth.day << "/" << hash\_lst[i].list[j].birth.mon << "/" << hash\_lst[i].list[j].birth.yr << endl;

cout << "Номер телефона:\t" << hash\_lst[i].list[j].phone << endl;

cout << "Hash:\t" << hash\_lst[i].key << endl;

cout << endl;

}

}

}

void FillHashTable(vector<Human> lst, vector<HashTable>& hash\_lst)

{

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

{

HashTable tmp;

tmp.HashFunc(lst[i].name, lst.size());

tmp.Add(lst[i]);

hash\_lst.push\_back(tmp);

}

Sort(hash\_lst);

int len = hash\_lst.size();

for (int i = 0; i < len - 1;)

{

if (hash\_lst[i].key == hash\_lst[i + 1].key)

{

hash\_lst[i].list.push\_back(hash\_lst[i + 1].list[0]);

hash\_lst.erase(hash\_lst.begin() + i + 1);

len--;

}

else i++;

}

}

void FindByName(vector<HashTable> hash\_lst, string key, int size)

{

int sum = 0;

for (int i = 0; i < key.length(); i++)

{

sum += abs(int(key[i]));

}

int key\_hash = sum % size;

for (int i = 0; i < hash\_lst.size(); i++)

{

if (hash\_lst[i].key == key\_hash)

{

for (int j = 0; j < hash\_lst[i].list.size(); j++)

{

if (key == hash\_lst[i].list[j].name)

{

cout << "Элемент найден!\n";

cout << "ФИО:\t" << hash\_lst[i].list[j].name << endl;

cout << "Дата рождения:\t" << hash\_lst[i].list[j].birth.day << "/" << hash\_lst[i].list[j].birth.mon << "/" << hash\_lst[i].list[j].birth.yr << endl;

cout << "Номер телефона:\t" << hash\_lst[i].list[j].phone;

cout << "Hash:\t" << hash\_lst[i].key << endl;

return;

}

}

cout << "===========================================================\n";

}

}

cout << "Элементов не найдено.\n";

}

void FindByHash(vector<HashTable> hash\_lst, int key)

{

for (int i = 0; i < hash\_lst.size(); i++)

{

if (key == hash\_lst[i].key)

{

cout << "Элемент(ы) с таким хешем найден(ы):\n";

PrintArr(hash\_lst[i].list);

cout << "===========================================================\n";

return;

}

}

cout << "Элементов не найдено.\n";

}

int main()

{

setlocale(LC\_ALL, "Russian");

srand(time(NULL));

int n;

vector<Human> list = CreateArr();

vector<HashTable> hash\_list;

PrintArr(list);

FillHashTable(list, hash\_list);

cout << "===========================================================\n\n";

cout << "Таблица хешей:\n";

PrintHashTable(hash\_list);

cout << "===========================================================\n\n";

cout << "===========================================================\n\n";

cout << "Методы поиска:\n";

cout << "| 1. Поиск по имени.\n";

cout << "| 2. Поиск по хешу.\n";

cout << "Выберите метод поиска (1-2): ";

cin >> n;

cin.get();

switch (n)

{

case 1:

{

string key;

cout << "Введите ФИО для поиска: ";

getline(cin, key);

while (key == "" || key[0] == ' ')

{

cout << "\nФИО введено некорректно.\n";

cout << "Введите ФИО для поиска: ";

getline(cin, key);

}

cout << "===========================================================\n\n";

FindByName(hash\_list, key, list.size());

break;

}

case 2:

{

int key;

cout << "Введите хеш для поиска: ";

cin >> key;

cout << "===========================================================\n\n";

FindByHash(hash\_list, key);

break;

}

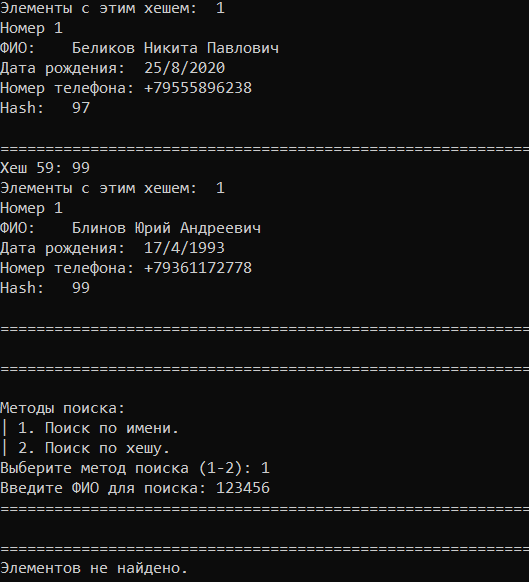
default:

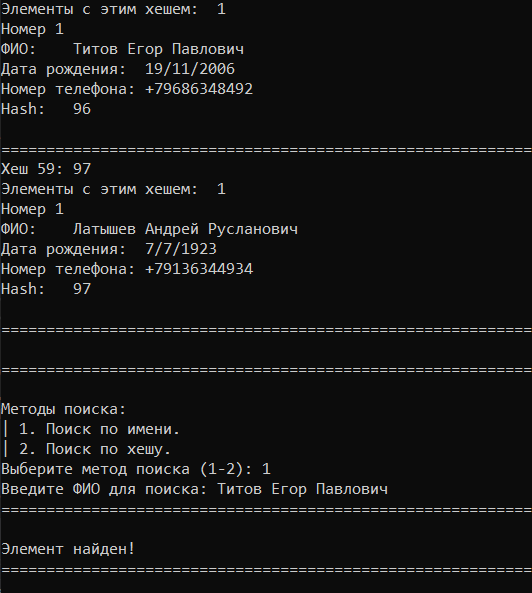
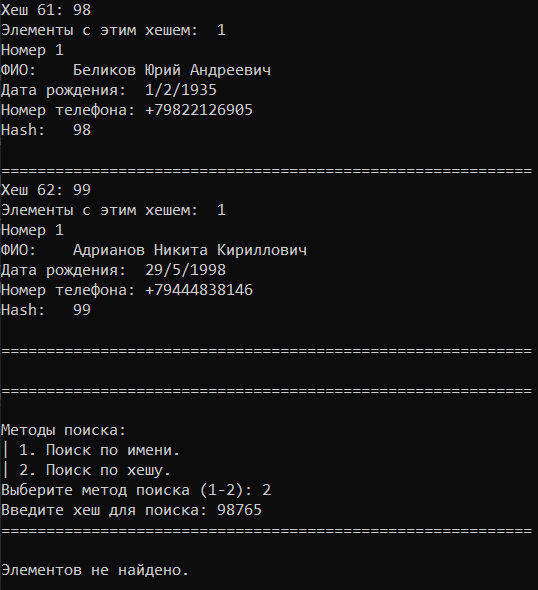
{

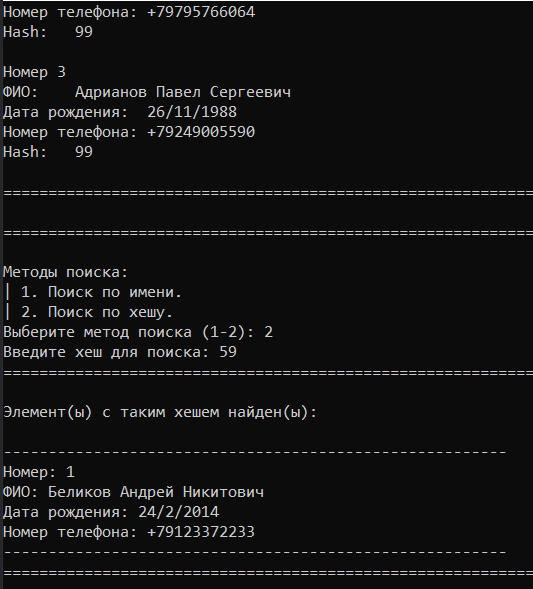
cout << "Ввод некорректен. Завершение работы...\n";

}

}

}**Скриншоты результатов работы программы**

****

****