Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет», ПНИПУ

ЛАБОРАТОРНАЯ РАБОТА

ХЭШ ТАБЛИЦЫ

Выполнил: студент группы РИС-23-3б

Артем Владимирович Швецов

Проверила: доцент кафедры ИТАС

Ольга Андреевна Полякова

Пермь 2024

**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.
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.

Для варианта 20:

|  |  |  |  |
| --- | --- | --- | --- |
| ФИО, дата\_рождения, №паспорта, | дата\_рождения, | H(k)= [M (kAmod1)], 0<A<1, mod1 – получение дробной части, [] – получение целой части | Метод цепочек |

**Анализ задачи**

1. Для получения хэш-кода создается функция, преобразующая строковое значение в целочисленное.
2. Навигация в хэш-таблице производится либо с помощью ключей, либо по порядку с пропуском пустых указателей.
3. Для регистрации коллизий объявляется глобальная переменная отслеживающая коллизии.

**Блок-схема**

**Код**

CommonFunc.h

#pragma once

#include <string>

#define \_USE\_MATH\_DEFINES

#include <math.h>

const double A = M\_PI\_4;

double mod1(double k) {

int intValue = static\_cast<int>(k);

return k - intValue;

}

int getHash(double k) {

return static\_cast<int> (M \* mod1(A \* k));

}

int getHash(std::string line) {

int hash = 0;

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

hash += static\_cast<int>(abs(line[i]) \* M\_LN10 + pow(line[i], 2) \* M\_PI\_2);

}

return getHash(abs(hash));

}

std::string fName[] = {

"fA", "fB", "fC", "fD", "fE", "fF", "fG", "fH", "fI", "fJ"

};

std::string iName[] = {

"iA", "iB", "iC", "iD", "iE", "iF", "iG", "iH", "iI", "iJ"

};

std::string oName[] = {

"oA", "oB", "oC", "oD", "oE", "oF", "oG", "oH", "oI", "oJ"

};

std::string makeName() {

return fName[rand() % 10] + ' ' + iName[rand() % 10] + ' ' + oName[rand() % 10];

}

std::string correctStr(int n, int size) {

std::string strn = std::to\_string(n);

while (strn.size() < size) {

strn = '0' + strn;

}

while (strn.size() > size) {

strn.erase(0, 1);

}

return strn;

}

std::string makeDate() {

return correctStr(rand() % 28 + 1, 2) + '/' + correctStr(rand() % 12 + 1, 2) + '/' + std::to\_string(rand() % 74 + 1950);

}

std::string makePass() {

return correctStr(rand() % 10000, 4) + correctStr(rand() % 1000000 \* 100 + rand(), 6);

}

LinearTable.h

#pragma once

#include <string>

struct NodeLinear

{

std::string key = "";

std::string dat = "";

};

struct HashTableLinear

{

NodeLinear\* table[M];

HashTableLinear() {

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

table[i] = nullptr;

}

}

};

LinearFunc.h

#pragma once

#include <fstream>

#include "CommonFunc.h"

#include "LinearTable.h"

bool add(HashTableLinear& table, std::string key, std::string custData) {

NodeLinear\* newNode = new NodeLinear;

newNode->key = key;

newNode->dat = custData;

int hash = getHash(key);

if (table.table[hash] == nullptr) {

table.table[hash] = newNode;

return true;

}

else {

int index = 0;

for (; table.table[(index + hash) % M] != nullptr && index < M; ++index);

collisionCount += index;

if (index != M) {

table.table[(index + hash) % M] = newNode;

return true;

}

else {

return false;

}

}

}

void makeHum(HashTableLinear& table) {

std::string dat = makeDate();

std::string name = makeName() + '|' + dat + '|' + makePass();

add(table, dat, name);

}

bool removeByKey(HashTableLinear& table, std::string key) {

int hash = getHash(key);

int index = 0;

bool isFind = false;

while (!isFind && index < M) {

if (table.table[(index + hash) % M] != nullptr) {

if (table.table[(index + hash) % M]->key != key) {

++index;

}

else {

isFind = true;

}

}

else {

++index;

}

}

if (index != M) {

delete table.table[(index + hash) % M];

table.table[(index + hash) % M] = nullptr;

return true;

}

else {

return false;

}

}

bool removeByData(HashTableLinear& table, std::string dat) {

int index = 0;

bool isFind = false;

while (!isFind && index < M) {

if (table.table[index] != nullptr) {

if (table.table[index]->dat != dat) {

++index;

}

else {

isFind = true;

}

}

else {

++index;

}

}

if (index != M) {

delete table.table[(index) % M];

table.table[(index) % M] = nullptr;

return true;

}

else {

return false;

}

}

NodeLinear\* get(HashTableLinear& table, std::string key) {

int hash = getHash(key);

int index = 0;

bool isFind = false;

while (!isFind && index < M) {

if (table.table[(index + hash) % M] != nullptr) {

if (table.table[(index + hash) % M]->key != key) {

++index;

}

else {

isFind = true;

}

}

else {

++index;

}

}

if (index != M) {

return table.table[(index + hash) % M];

}

else {

return nullptr;

}

}

void printTable(HashTableLinear& table) {

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

if (table.table[i] != nullptr) {

NodeLinear\* current = table.table[i];

std::cout << current->key << " [" << current->dat << "]" << std::endl;

}

}

std::cout << std::endl;

}

void saveTable(HashTableLinear& table) {

std::ofstream out;

out.open("File.txt");

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

if (table.table[i] != nullptr) {

NodeLinear\* current = table.table[i];

out << current->key << std::endl << current->dat << std::endl;

}

}

out.close();

}

void clearTable(HashTableLinear& table) {

collisionCount = 0;

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

NodeLinear\* current = table.table[i];

if (table.table[i] != nullptr) {

delete current;

table.table[i] = nullptr;

}

}

}

void readTable(HashTableLinear& table) {

clearTable(table);

std::ifstream in;

in.open("File.txt");

while (!in.eof()) {

std::string key;

getline(in, key);

std::string data;

getline(in, data);

if (key != "" && data != "") {

add(table, key, data);

}

}

in.close();

}

ChainTable.h

#pragma once

#include <string>

struct NodeChain

{

std::string key = "";

std::string dat = "";

NodeChain\* ptrNext = nullptr;

NodeChain\* ptrPrev = nullptr;

};

struct HashTableChain

{

NodeChain\* table[M];

HashTableChain() {

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

table[i] = nullptr;

}

}

};

ChainFunc.h

#pragma once

#include <fstream>

#include "CommonFunc.h"

#include "ChainTable.h"

bool add(HashTableChain& table, std::string key, std::string custData) {

NodeChain\* newNode = new NodeChain;

newNode->key = key;

newNode->dat = custData;

int hash = getHash(key);

if (table.table[hash] == nullptr) {

table.table[hash] = newNode;

return true;

}

else {

NodeChain\* current = table.table[hash];

while (current->ptrNext != nullptr) {

current = current->ptrNext;

}

current->ptrNext = newNode;

newNode->ptrPrev = current;

collisionCount++;

return true;

}

}

void makeHum(HashTableChain& table) {

std::string dat = makeDate();

std::string name = makeName() + '|' + dat + '|' + makePass();

add(table, dat, name);

}

bool removeByKey(HashTableChain& table, std::string key) {

int hash = getHash(key);

NodeChain\* current = table.table[hash];

while (current != nullptr) {

if (current->key == key) {

if (current->ptrPrev != nullptr) {

current->ptrPrev->ptrNext = current->ptrNext;

}

else {

table.table[hash] = current->ptrNext;

}

if (current->ptrNext != nullptr) {

current->ptrNext->ptrPrev = current->ptrPrev;

}

delete current;

return true;

}

current = current->ptrNext;

}

return false;

}

bool removeByData(HashTableChain& table, std::string dat) {

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

{

NodeChain\* current = table.table[i];

while (current != nullptr) {

if (current->dat == dat) {

if (current->ptrPrev != nullptr) {

current->ptrPrev->ptrNext = current->ptrNext;

}

else {

table.table[i] = current->ptrNext;

}

if (current->ptrNext != nullptr) {

current->ptrNext->ptrPrev = current->ptrPrev;

}

delete current;

return true;

}

current = current->ptrNext;

}

}

return false;

}

NodeChain\* get(HashTableChain& table, std::string key) {

int hash = getHash(key);

NodeChain\* current = table.table[hash];

while (current != nullptr)

{

if (current->key == key) {

return current;

}

current = current->ptrNext;

}

return nullptr;

}

void printTable(HashTableChain& table) {

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

NodeChain\* current = table.table[i];

while (current != nullptr)

{

std::cout << current->key << " [" << current->dat << "]" << std::endl;

current = current->ptrNext;

}

}

std::cout << std::endl;

}

void saveTable(HashTableChain& table) {

std::ofstream out;

out.open("File.txt");

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

NodeChain\* current = table.table[i];

while (current != nullptr)

{

out << current->key << std::endl << current->dat << std::endl;

current = current->ptrNext;

}

}

out.close();

}

void clearTable(HashTableChain& table) {

collisionCount = 0;

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

NodeChain\* current = table.table[i];

while (current != nullptr)

{

table.table[i] = current->ptrNext;

delete current;

current = table.table[i];

}

}

}

void readTable(HashTableChain& table) {

clearTable(table);

std::ifstream in;

in.open("File.txt");

while (!in.eof()) {

std::string key;

getline(in, key);

std::string data;

getline(in, data);

if (key != "" && data != "") {

add(table, key, data);

}

}

in.close();

}

Main.cpp

#include <iostream>

#include <string>

extern const int M = 90;

extern int collisionCount = 0;

#include "LinearFunc.h"

#include "ChainFunc.h"

using namespace std;

int main() {

system("chcp 1251 > null");

int navig, subnavig, count = 100;

int tableMode = 0; // 1 - Linear, 2 - Chain

string buffer;

bool flag;

HashTableLinear tableL;

HashTableChain tableC;

do {

cout << "1. Создание таблицы" << endl;

cout << "2. Просмотр таблицы" << endl;

cout << "3. Поиск элемента" << endl;

cout << "4. Добавление элемента" << endl;

cout << "5. Удаление элемента" << endl;

cout << "6. Сохранение таблицы в файле" << endl;

cout << "7. Чтение таблицы из файла" << endl;

cout << "8. Отчистка таблицы" << endl;

cout << "0. Завершение программы" << endl;

cout << endl << ">";

cin >> navig;

cin.ignore();

switch (navig)

{

case 1:

cout << "Выберете тип таблицы: 1. Линейный 2. Цепной" << endl;

cout << endl << ">";

cin >> subnavig;

cin.ignore();

if (subnavig == 1) {

tableMode = 1;

clearTable(tableL);

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

makeHum(tableL);

}

cout << "Сгенерирована hash-таблица" << endl;

}

else if (subnavig == 2)

{

clearTable(tableC);

tableMode = 2;

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

makeHum(tableC);

}

cout << "Сгенерирована hash-таблица" << endl;

}

else {

cout << "Некорректный индекс" << endl;

}

cout << endl;

break;

case 2:

if (tableMode == 1) {

printTable(tableL);

cout << "Число коллизий: " << collisionCount << endl;

}

else if (tableMode == 2) {

printTable(tableC);

cout << "Число коллизий: " << collisionCount << endl;

}

else {

cout << "Таблица отсутствует" << endl;

}

cout << endl;

break;

case 3:

if(!tableMode) {

cout << "Таблица отсутствует" << endl << endl;

break;

}

cout << "Введите ключ поиска: ";

getline(cin, buffer);

flag = false;

if (tableMode == 1) {

NodeLinear\* node = get(tableL, buffer);

flag = node != nullptr;

if (flag) buffer = node->dat;

}

else if (tableMode == 2) {

NodeChain\* node = get(tableC, buffer);

flag = node != nullptr;

if (flag) buffer = node->dat;

}

if (flag) {

cout << "Элемент найден: " << buffer << endl;

}

else {

cout << "Элемент с заданным ключом не найден" << endl;

}

cout << endl;

break;

case 4:

if (!tableMode) {

cout << "Таблица отсутствует" << endl << endl;

break;

}

cout << "Выберете режим добавления: 1. ДСЧ 2. Ручной" << endl;

cout << endl << ">";

cin >> subnavig;

if (subnavig == 1) {

if (tableMode == 1) {

makeHum(tableL);

cout << "Элемент добавлен";

}

else if (tableMode == 2) {

makeHum(tableC);

cout << "Элемент добавлен";

}

}

if (subnavig == 2) {

string s1;

string s2;

string s3;

string custData;

cout << "Введите через пробел: дата рождения, ФИО, номер паспорта" << endl;

cin >> s1;

cin >> s2;

cin >> s3;

custData = s2 + '|' + s1 + '|' + s3;

if (tableMode == 1) {

add(tableL, s1, custData);

cout << "Элемент добавлен";

}

else if (tableMode == 2) {

add(tableC, s1, custData);

cout << "Элемент добавлен";

}

}

else {

cout << "Некорректный индекс" << endl;

}

cout << endl;

break;

case 5:

if (!tableMode) {

cout << "Таблица отсутствует" << endl << endl;

break;

}

cout << "Выберете режим удаления: 1. По ключу 2. По данным" << endl;

cout << endl << ">";

cin >> subnavig;

cin.ignore();

flag = false;

if (subnavig == 1) {

cout << "Введите ключ для удаления: ";

getline(cin, buffer);

if (tableMode == 1) {

flag = removeByKey(tableL, buffer);

}

else if (tableMode == 2) {

flag = removeByKey(tableC, buffer);

}

if (flag) {

cout << "Элемент с заданным ключом удален" << endl;

}

else {

cout << "Элемент с заданным ключом не найден" << endl;

}

}

else if (subnavig == 2) {

cout << "Введите данные для удаления: ";

getline(cin, buffer);

if (tableMode == 1) {

flag = removeByData(tableL, buffer);

}

else if (tableMode == 2) {

flag = removeByData(tableC, buffer);

}

if (flag) {

cout << "Элемент с заданными данными удален" << endl;

}

else{

cout << "Элемент с заданными данными не найден" << endl;

}

}

else {

cout << "Некорректный индекс" << endl;

}

cout << endl;

break;

case 6:

if (tableMode == 1) {

saveTable(tableL);

cout << "Таблица сохранена" << endl;

}

else if (tableMode == 2) {

saveTable(tableC);

cout << "Таблица сохранена" << endl;

}

else {

cout << "Таблица отсутствует" << endl;

}

cout << endl;

break;

case 7:

cout << "Выберете тип таблицы: 1. Линейный 2. Цепной" << endl;

cout << endl << ">";

cin >> subnavig;

cin.ignore();

if (subnavig == 1) {

tableMode = 1;

readTable(tableL);

cout << "Востановлена hash-таблица" << endl;

}

else if (subnavig == 2)

{

tableMode = 2;

readTable(tableC);

cout << "Востановлена hash-таблица" << endl;

}

else {

cout << "Некорректный индекс" << endl;

}

cout << endl;

break;

case 8:

if (tableMode == 1) {

clearTable(tableL);

cout << "Таблица отчищена" << endl;

}

else if (tableMode == 2) {

clearTable(tableC);

cout << "Таблица отчищена" << endl;

}

else {

cout << "Таблица отсутствует" << endl;

}

cout << endl;

break;

default:

break;

}

} while (navig != 0);

return 0;

}

**Решение**

**Выводы**

В ходе выполнения лабораторной работы были изучены хэш таблицы и механизмы их работы.

**Github**

<https://github.com/Hitikov/Lab_HashTables>