Пермский национальный исследовательский политехнический университет.

Лабораторная работа.

«Хэш таблицы».

Выполнил: студент группы РИС-23-2б

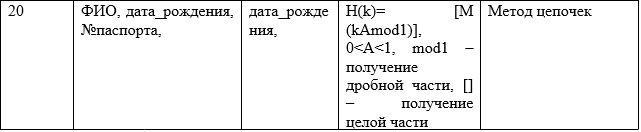
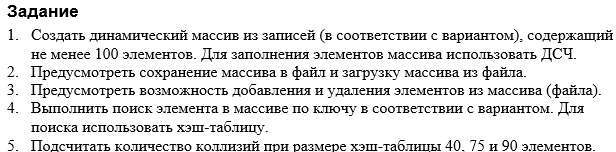
Вековшинин Иван Николаевич

Проверила: доцент кафедры ИТАС

О.А. Полякова.

2024 г.

Постановка задачи:



Код:

#include <iostream>

#include <Windows.h>

#include <vector>

#include <string>

#include <cmath>

#include <ctime>

using namespace std;

const int M = 5;

const double A = 3.141592653589793;

int collisionCounter = 0;

struct Node {

string key = "";

string value = "";

Node\* next = nullptr;

Node\* prev = nullptr;

};

struct HashTable {

Node\* table[M];

HashTable() {

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

table[i] = nullptr;

}

}

};

double mod1(double k) {

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

return k - intPart;

}

int getHash(double k) {

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

}

int getHash(string line) {

int n = 0;

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

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

}

return getHash(abs(n));

}

bool add(HashTable& table, string key, string elem) {

Node\* newNode = new Node;

newNode->key = key;

newNode->value = elem;

newNode->next = nullptr;

newNode->prev = nullptr;

int hash = getHash(key);

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

table.table[hash] = newNode;

return true;

}

else {

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

while (current->next != nullptr) {

current = current->next;

}

current->next = newNode;

newNode->prev = current;

collisionCounter++;

return true;

}

}

bool removeByKey(HashTable& table, string key) {

int hash = getHash(key);

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

while (current != nullptr) {

if (current->key == key) {

if (current->prev != nullptr) {

current->prev->next = current->next;

}

else {

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

}

if (current->next != nullptr) {

current->next->prev = current->prev;

}

delete current;

return true;

}

current = current->next;

}

return false;

}

bool removeByValue(HashTable& table, string elem) {

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

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

while (current != nullptr) {

if (current->value == elem) {

if (current->prev != nullptr) {

current->prev->next = current->next;

}

else {

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

}

if (current->next != nullptr) {

current->next->prev = current->prev;

}

delete current;

return true;

}

current = current->next;

}

}

return false;

}

Node\* get(HashTable& table, string key) {

int hash = getHash(key);

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

return current;

}

void print(HashTable& table) {

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

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

while (current != nullptr) {

cout << "[" << current->key << ": " << current->value << "]\n";

current = current->next;

}

}

cout << endl;

}

string surnames[] = {

"Vlasov", "Vekovshinin", "Stalin", "Mejron", "Stasov", "Popov"

};

string names[] = {

"Vlad", "Stas", "Sanya", "Danya", "Dyna", "Vany"

};

string patronymics[] = {

"Vlasov123", "Vekovshinin123", "Stal123in", "Mejron243", "Stas2143ov", "A234OAOAOAOAO"

};

string generateFullName() {

return surnames[rand() % 6] + ' ' + names[rand() % 6] + ' ' + patronymics[rand() % 6];

}

string correctStr(int n, int len) {

string strn = to\_string(n);

while (strn.size() < len) {

strn = '0' + strn;

}

while (strn.size() > len) {

strn.erase(0, 1);

}

return strn;

}

string generateBirthday() {

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

}

string generatePassportNum() {

return correctStr(rand() % 10000, 4) + ' ' + correctStr((rand() % 100000000 \* 100 + rand()), 6);

}

int main() {

setlocale(LC\_ALL, "Ru");

srand(time(NULL));

HashTable table;

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

string BirthDay = generateBirthday();

string fio = generateFullName();

string newHuman = fio + " | " + BirthDay + " | " + generatePassportNum();

add(table, fio, newHuman);

}

print(table);

int existingInd = rand() % M;

while (table.table[existingInd] == nullptr) {

existingInd = rand() % M;

}

Node\* randomHuman = table.table[existingInd];

string KeyToRemove = randomHuman->key;

cout << "Удаление по ключу:" << KeyToRemove << endl;

if (removeByKey(table, KeyToRemove)) {

cout << "Элемент с ключем удален" << endl;

}

else { cout << "Элемент с ключем не найден" << endl; }

print(table);

int existingInd2 = rand() % M;

while (table.table[existingInd2] == nullptr) {

existingInd2 = rand() % M;

}

Node\* randomHuman2 = table.table[existingInd2];

string valueToRemove = randomHuman2->value;

cout << "Удаление по значению:" << valueToRemove << endl;

if (removeByValue(table, KeyToRemove)) {

cout << "Элемент с значением удален" << endl;

}

else { cout << "Элемент с значением не найден" << endl; }

print(table);

int existingInd3 = rand() % M;

while (table.table[existingInd3] == nullptr) {

existingInd3 = rand() % M;

}

Node\* randomHuman3 = table.table[existingInd3];

string keyToGet = randomHuman3->key;

cout << endl << "Ключ: " << keyToGet << endl;

Node\* node = get(table, keyToGet);

if (node != nullptr) {

cout << "Найден!: " << node->value << endl;

}

else {

cout << "Не найден!" << endl;

}

cout << "Число коллизий: = " << collisionCounter;

}

Результат работы программы:

