Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №1**

По дисциплине: «Аппаратное обеспечение интеллектуальных систем»

Тема: «Применение хэширование для ассоциативного поиска по ключам»

**Выполнил:**

Студент 2 курса

Группы ИИ-23

Макаревич Н. Р.

**Проверил:**

Михно Е.В.

Брест 2024

**Цель работы:** изучить основные принципы хэширования, построить программную модель, обеспечивающую формирование хэш-таблиц и ассоциативный поиск по ключам.

**Задание:**

**1)** Написать на ЯВУ программу, осуществляющую просмотр, модификацию и сохранение таблицы записей. Реализовать добавление, удаление, редактирование записей и поиск по ключу перебором. Таблица должна содержать N строк (в таблицу заносится n записей) и не менее трех полей (с одним ключевым полем).

**2)** Написать на ЯВУ программу, которая преобразует таблицу данных из первого задания в хэш-таблицу на N строк (n записей). Для хэш-таблицы реализовать возможности добавления, удаления, модификации и поиска записей. Хэш-функция и метод обработки коллизий выбираются согласно варианту. Рассчитать вероятность возникновения коллизий (количество возникших коллизий при добавлении данных разделить на количество добавленных в хэш-таблицу записей) и среднее количество обращений к таблице во время поиска записи по ключу (сумма количества обращений к таблице при поиске каждой записи в хэш-таблице деленная на количество записей в хэш-таблице).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Вариант** | **Тип хэш-функции** | **Метод обработки коллизий** | **Количест-во записей n** | **Объем хэш-таблицы N** |
| 3 | Мультипликативный (1, 3, последний символы) + метод умножения | линейное зондирование | 40 | 1.1\*n |

Заголовочный файл задания 1:

#pragma once

#include <iostream>

#include <string>

#include <vector>

class BasicTable{

public:

struct Dog {

std::string name;

std::string breed;

int age;

Dog();

Dog(std::string name, std::string breed, int age);

};

BasicTable(int size);

void showTable();

void addData(std::string name, std::string breed, int age);

void deleteData(std::string key);

void redactData(std::string key, std::string newname, std::string newbreed, int newage);

int searchData(std::string key);

std::vector<Dog> passData();

private:

int N;

std::vector <Dog> table;

};

Исходный файл задания 1:

#include "BasicTable.h"

BasicTable::Dog::Dog() {

name = "NULL";

breed = "NULL";

age = NULL;

}

BasicTable::Dog::Dog(std::string newname, std::string newbreed, int newage) {

name = newname;

breed = newbreed;

age = newage;

}

BasicTable::BasicTable(int size) {

table = std::vector <Dog>(size, Dog());

N = size;

}

void BasicTable::showTable() {

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

std::cout << i << " " << table[i].name << " " << table[i].breed << " " << table[i].age << "\n";

}

}

void BasicTable::addData(std::string name, std::string breed, int age) {

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

if (table[i].name == "NULL") {

table[i] = Dog(name, breed, age);

return;

}

}

std::cout << "No more space.\n";

}

void BasicTable::deleteData(std::string key) {

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

if (table[i].name == key) {

table[i] = Dog();

}

}

}

void BasicTable::redactData(std::string key, std::string newname, std::string newbreed, int newage) {

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

if (table[i].name == key) {

table[i] = Dog(newname, newbreed, newage);

}

}

}

int BasicTable::searchData(std::string key) {

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

if (table[i].name == key) {

return i;

}

}

return NULL;

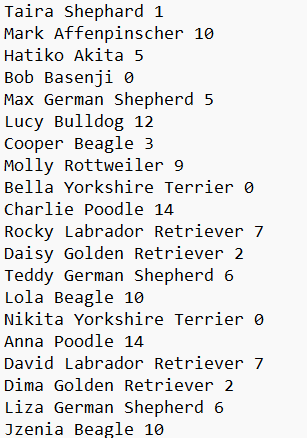
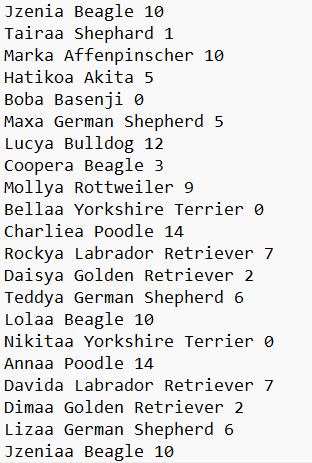
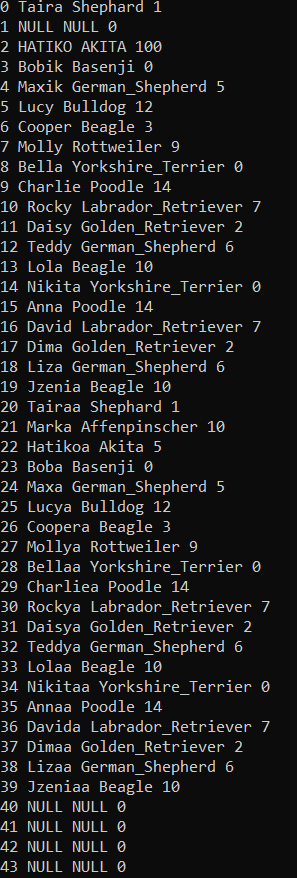
}

std::vector<BasicTable::Dog> BasicTable::passData() {

return table;

}

Исходная таблица:

Код взаимодействия для задания 1 и 2:

void loadFromFile(BasicTable &basicTable) {

std::ifstream fin("list.txt");

std::string name;

std::string breed;

int age;

for (int i = 0; i < 40; i++) {

fin >> name >> breed >> age;

basicTable.addData(name, breed, age);

}

fin.close();

}

int main()

{

BasicTable basicTable(44);

loadFromFile(basicTable);

std::cout << "Basic table:\n";

basicTable.showTable();

basicTable.deleteData("Mark");

basicTable.redactData("Hatiko", "HATIKO", "AKITA", 100);

int ind = basicTable.searchData("Lola");

if (ind)

std::cout << "\nFound at " << ind << "\n\n";

basicTable.showTable();

HashTable hashTable(44);

hashTable.getData(basicTable.passData());

std::cout << "\nHash table:\n";

hashTable.showTable();

hashTable.deleteData("Dima");

hashTable.redactData("Boba", "Boba", "SOBAKA", 999);

ind = hashTable.searchData("David");

if (ind)

std::cout << "\nFound at " << ind << "\n\n";

hashTable.showTable();

hashTable.chanceOfColision();

}

Заголовочный файл задания 2:

#pragma once

#include <iostream>

#include <string>

#include <vector>

#include "BasicTable.h"

class HashTable

{

public:

struct Dog {

std::string name;

std::string breed;

int age;

Dog();

Dog(std::string name, std::string breed, int age);

};

HashTable(int size);

void showTable();

void getData(std::vector<BasicTable::Dog> HashTable);

void addData(std::string name, std::string breed, int age);

void deleteData(std::string);

int searchData(std::string);

void redactData(std::string key, std::string name, std::string breed, int age);

void chanceOfColision();

private:

int N;

double numOfCollisions;

double numOfData;

double numOfSearchTries;

std::vector <bool> collisions;

std::vector <Dog> table;

int hashF(std::string key);

bool tryAdd(int ind, std::string name, std::string breed, int age);

};

Исходный файл задания 2:

#include "HashTable.h"

#include <cmath>

HashTable::Dog::Dog() {

name = "NULL";

breed = "NULL";

age = NULL;

}

HashTable::Dog::Dog(std::string newname, std::string newbreed, int newage) {

name = newname;

breed = newbreed;

age = newage;

}

HashTable::HashTable(int size) {

table = std::vector <Dog>(size, Dog());

collisions = std::vector <bool>(size, false);

N = size;

numOfCollisions = 0;

numOfData = 0;

numOfSearchTries = 0;

}

void HashTable::showTable() {

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

if (collisions[i])

std::cout << "\*";

std::cout << i << " " << table[i].name << " " << table[i].breed << " " << table[i].age << "\n";

}

}

int HashTable::hashF(std::string key) {

int firstAscii = static\_cast<int>(key[0]);

int thirdAscii = static\_cast<int>(key[2]);

int lastAscii = static\_cast<int>(key[key.length() - 1]);

double product = firstAscii \* thirdAscii \* lastAscii;

double C = 0.6180339887;

double normHashAdress = std::modf(product \* C, &normHashAdress);

return std::trunc(N \* normHashAdress);

}

void HashTable::addData(std::string name, std::string breed, int age) {

if (name != "NULL" || name != "") {

int index = hashF(name);

if (!tryAdd(index, name, breed, age)) {

numOfCollisions++;

collisions[index] = true;

}

}

}

bool HashTable::tryAdd(int ind, std::string name, std::string breed, int age) {

if (table[ind].name == "NULL") {

table[ind] = Dog(name, breed, age);

numOfData++;

return true;

}

else if (ind + 1 == N)

return true;

else {

for (int i = ind + 1; i < N; i++) {

if (table[i].name == "NULL") {

table[i] = Dog(name, breed, age);

numOfData++;

return false;

}

}

}

return true;

}

void HashTable::getData(std::vector<BasicTable::Dog> basicTable) {

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

addData(basicTable[i].name, basicTable[i].breed, basicTable[i].age);

}

}

int HashTable::searchData(std::string key) {

numOfSearchTries++;

int ind = hashF(key);

if (table[ind].name == key) {

return ind;

}

else {

for (int i = ind + 1; i < N; i++) {

numOfSearchTries++;

if (table[i].name == key) {

return i;

}

}

}

return -1;

}

void HashTable::deleteData(std::string key) {

int ind = searchData(key);

if (ind) {

table[ind] = Dog();

numOfData--;

}

}

void HashTable::redactData(std::string key, std::string name, std::string breed, int age) {

int ind = searchData(key);

if (ind) {

table[ind] = Dog(name, breed, age);

}

}

void HashTable::chanceOfColision() {

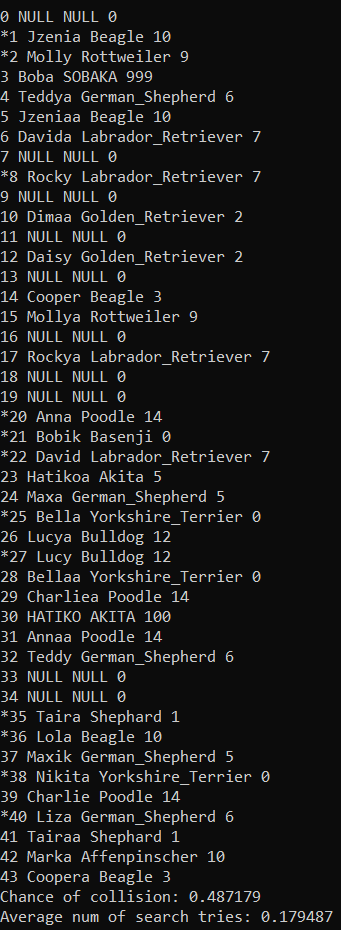
std::cout << "Chance of collision: " << numOfCollisions / numOfData << "\n";

std::cout << "Average num of search tries: " << numOfSearchTries / numOfData << "\n";

numOfSearchTries = 0;

}

Хеш-таблица:

Расчетные данные:



**Вывод:** изучил основные принципы хэширования, построил программную модель, обеспечивающую формирование хэш-таблиц и ассоциативный поиск по ключам.