

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**«Дальневосточный федеральный университет»**

(ДВФУ)

**институт математики и компьютерных технологий**

**Департамент информационных и компьютерных систем**

**ОТЧЕТ**

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

по теме «Хэширование»

**направление подготовки**

**09.03.03 Прикладная информатика**

**Прикладная информатика в экономике**

|  |  |
| --- | --- |
| Выполнил студент гр. Б9121 / 4 | |
|  |  |
|  | |
| Проверил ст. преподаватель | |
|  | Елсукова Е. А. |
|  | |
| не зачтено/зачтено | |

г. Владивосток

2022г.

Содержание

[1 Условие задачи 3](#_Toc125548209)

[2 Функциональное описание 4](#_Toc125548210)

[2.1 Хэш-функции 4](#_Toc125548211)

[2.2 Коллизии и их решение 5](#_Toc125548212)

[2.3 Интерфейс программы 6](#_Toc125548213)

[3 Полученные результаты 9](#_Toc125548214)

[Приложение 10](#_Toc125548215)

1 Условие задачи

Программно реализовать поиск с использованием хеширования:

1. Разработать две хэш-функции, предусмотреть обработку коллизий. Хеш–функции и метод для разрешения коллизий выбрать любые из указанных в теоретической части.
2. Захэшировать входные данные; входные данные – массив записей. Структура записей по вариантам используется из лабораторной работы «Алгоритмы внутренней сортировки». Оценить реализованные хэш-функции на число коллизий в исходном наборе данных и далее использовать хэш-функцию, которая минимизирует коллизии.
3. На основании полученных хэш-значений построить хэш-таблицу.
4. Реализовать Поиск, Добавление, Удаление элементов.

2 Функциональное описание

2.1 Хэш-функции

Данные были захешированы двумя методами: деления, умножения.

Хеш-функция, основанная на методе деления, определяется формулой , где *k* – ключ, *n* – простое число, *m* – размер таблица. При этом, чем больше простое число, тем больше тем меньше коллизий (см. рисунок 1).

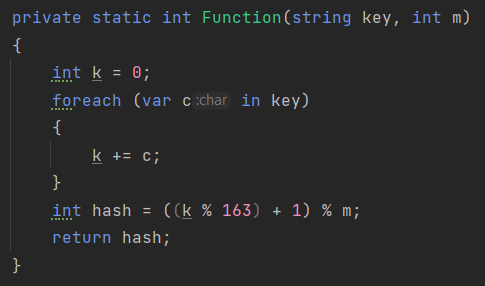


Рисунок 1 – Реализация метода деления

Хеш-функция, основанная на методе умножения, определяется формулой , где - скобки взятия целой части, - скобки взятия дробной части, - скобки приоритета, *k* – ключ, *m* – размер таблицы, *A* – рациональное число, заданное золотым сечением (см. рисунок 2).

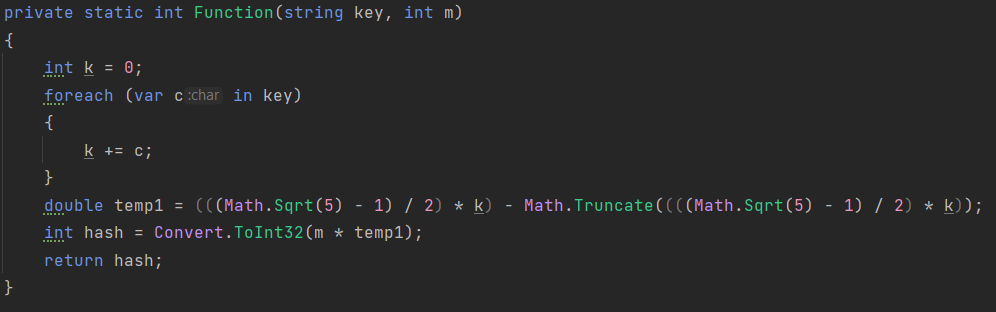


Рисунок 2 – Реализация метода умножения

Выбор вышеописанных метод оправдан простотой их реализации и тем, что названия методов являются антонимами.

2.2 Коллизии и их решение

Для разрешения коллизий выбран метод внутренней цепочки, реализованный с помощью структуры – список, который хранит список идентичных хеш-значений.



Рисунок 3 – Реализация метода внутренних цепочек

При сравнении хеш-функций видно, что при хешировании данных методом умножения получено меньше коллизий: 4 vs 2 (см. рисунки 4, 5).

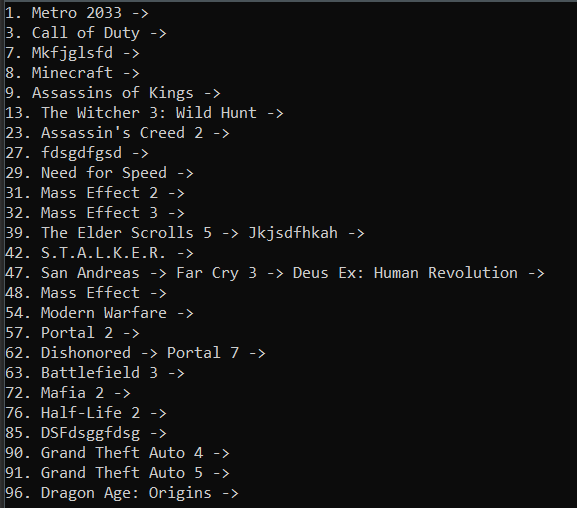


Рисунок 4 – Таблица при методе деления

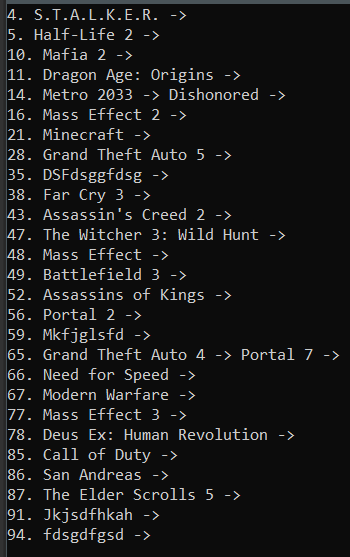


Рисунок 5 – Таблица при методе умножения

2.3 Интерфейс программы

Первый уровень интерфейса (см. рисунок 6).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 6 – Первый уровень интерфейса.

Визуализированы таблица хеширования и основные опции для каждой функции (см. рисунки 7, 8).

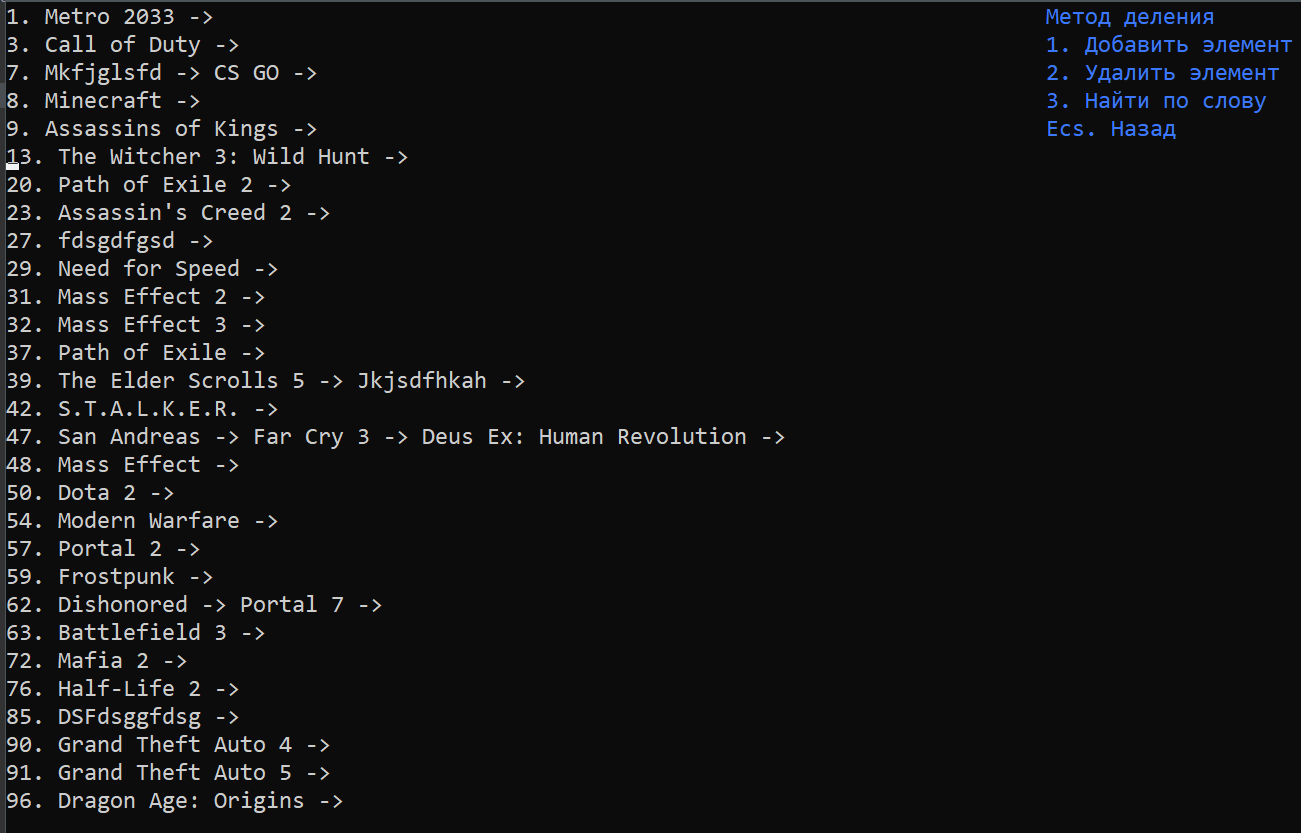


Рисунок 7 – Интерфейс хеш-функции методом деления

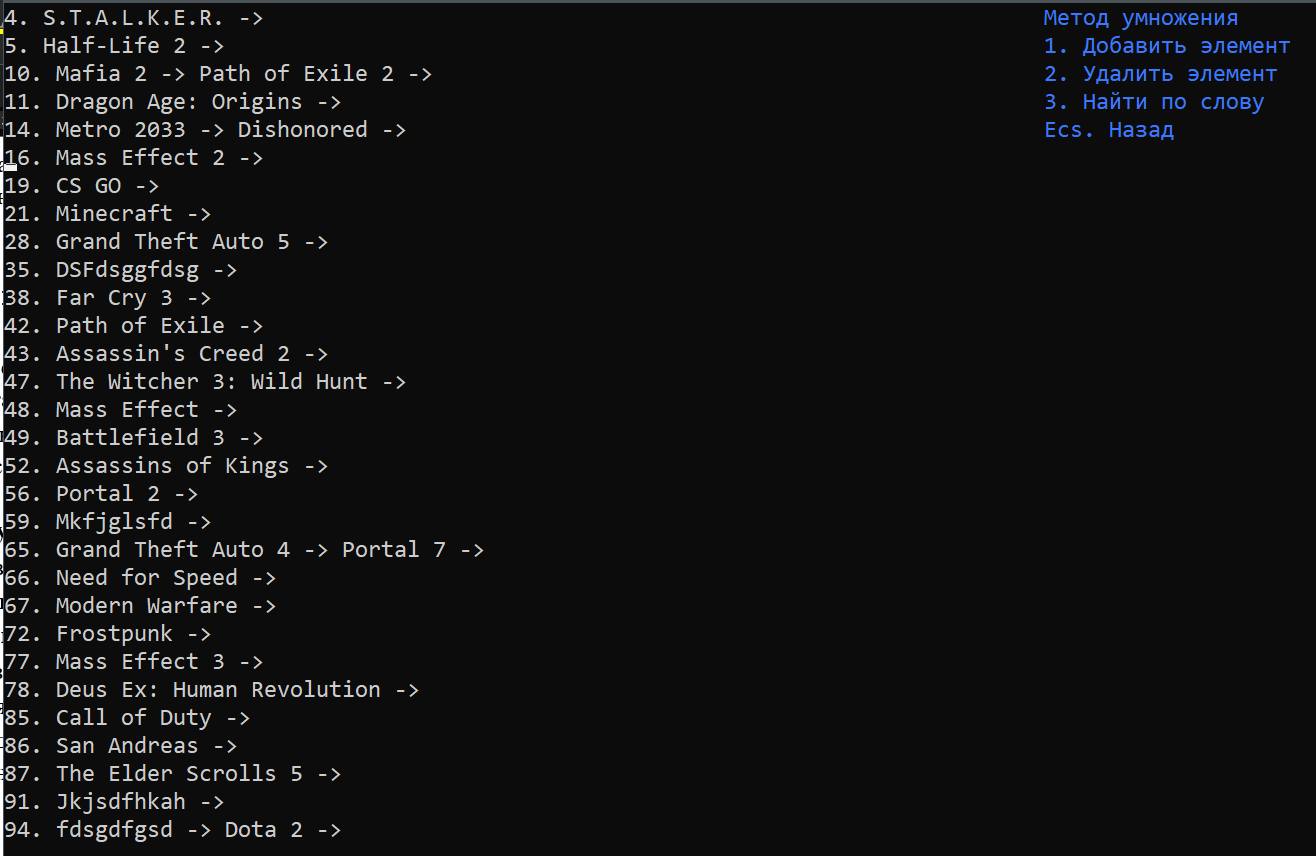


Рисунок 8 – Интерфейс хеш-функции методом умножения

При вызове опции «Добавить элемент». Появляется программное уведомление о необходимости ввести имя объекта. После ввода оно помещается в таблицу хеширования (см. рисунок 9, 10).

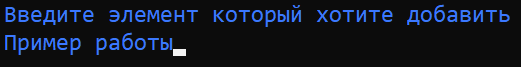


Рисунок 9 – Добавление элемента



Рисунок 10 – Результат добавления

При вызове вызови опции «Удалить элемент». Появляется программное уведомление о необходимости ввести имя объекта. После ввода оно удаляется из таблицы хеширования (см. рисунок 11).

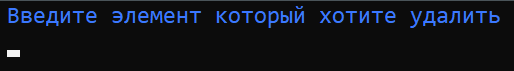


Рисунок 11 – Удаление элемента

При вызове вызови опции «Найти по слову». Появляется программное уведомление о необходимости ввести имя объекта. После ввода появляется хеш-значение объекта (см. рисунок 12).



Рисунок 12 – Результат поиска

3 Полученные результаты

В результате выполнения лабораторной работы были рассмотрены две хеш-функции, а именно метод деления и м метод умножения, и доказано, что наиболее эффективно распределяет значения объектов в хеш-таблицы хеш-функция методом умножения. Он вывел меньшее количество коллизий.

Также был реализовано разрешение коллизий методом внутренней цепочки через список списков, что позволило сохранять в хеш-таблице разные объекты с одинаковым хеш-значением.

Приложение

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Hashing

{

internal class Program

{

static void OptopnAddM(ref Data data)

{

data.Print();

Console.ForegroundColor = ConsoleColor.Blue;

Console.SetCursorPosition(50, 0);

Console.WriteLine("Введите элемент который хотите добавить");

Console.SetCursorPosition(50, 1);

string str = Console.ReadLine();

Divisions.Add(ref data, str);

StreamWriter sw = new StreamWriter("Data.txt", true);

sw.WriteLine(str);

sw.Close();

Console.ResetColor();

}

static void OptiondDeleteM(ref Data data)

{

data.Print();

Console.ForegroundColor = ConsoleColor.Blue;

Console.SetCursorPosition(50, 0);

Console.WriteLine( "Введите элемент который хотите удалить");

Console.SetCursorPosition(50, 1);

string tmpstr = Console.ReadLine();

Console.SetCursorPosition(50, 1);

if (Divisions.CanDelete(ref data,tmpstr))

Console.WriteLine("Элемент удалён ");

else

Console.WriteLine("Такого элемента нет в хеш-таблице");

Console.ResetColor();

Console.ReadKey();

}

static void OptionSearchM(ref Data data)

{

data.Print();

Console.ForegroundColor = ConsoleColor.Blue;

Console.SetCursorPosition(50, 0);

Console.WriteLine("Введите слово которое хотите найти");

Console.SetCursorPosition(50, 1);

string str = Console.ReadLine();

Console.SetCursorPosition(50, 1);

if (Divisions.Search(ref data, str) == -1)

Console.WriteLine("Неверно введены данные");

else

Console.WriteLine(Divisions.Search(ref data, str) + ". " + str);

Console.ResetColor();

Console.ReadKey();

}

static void OptionAddD(ref Data data)

{

data.Print();

Console.ForegroundColor = ConsoleColor.Blue;

Console.SetCursorPosition(50, 0);

Console.WriteLine("Введите элемент который хотите добавить");

Console.SetCursorPosition(50, 1);

string str = Console.ReadLine();

Multiplications.Add(ref data, str);

Console.ResetColor();

StreamWriter sw = new StreamWriter("Data.txt", true);

sw.WriteLine(str);

sw.Close();

Console.ResetColor();

}

static void OptionDeleteD(ref Data data)

{

data.Print();

Console.ForegroundColor = ConsoleColor.Blue;

Console.SetCursorPosition(50, 0);

Console.WriteLine("Введите элемент который хотите удалить");

Console.SetCursorPosition(50, 1);

string tmpstr = Console.ReadLine();

Console.SetCursorPosition(50, 1);

if (Multiplications.CanDelete(ref data, tmpstr))

Console.WriteLine("Элемент удалён ");

else

Console.WriteLine("Такого элемента нет в хеш-таблице");

Console.ResetColor();

Console.ReadKey();

}

static void OptionSearchD(ref Data data)

{

data.Print();

Console.ForegroundColor = ConsoleColor.Blue;

Console.SetCursorPosition(50, 0);

Console.WriteLine("Введите слово которое хотите найти");

Console.SetCursorPosition(50, 1);

string str = Console.ReadLine();

Console.SetCursorPosition(50, 1);

if (Multiplications.Search(ref data, str) == -1)

Console.WriteLine("Неверно введены данные");

else

Console.WriteLine(Multiplications.Search(ref data, str) + ". " + str);

Console.ResetColor();

Console.ReadKey();

}

static void OptionsD()

{

Data data = new Data(99);

StreamReader stream = new StreamReader("Data.txt");

string str;

while((str = stream.ReadLine())!=null)

{

Divisions.Add(ref data, str);

}

stream.Close();

bool exit = true;

do

{

Console.Clear();

data.Print();

Console.ForegroundColor = ConsoleColor.Blue;

Console.SetCursorPosition(80, 0);

Console.WriteLine("Метод деления");

Console.SetCursorPosition(80, 1);

Console.WriteLine("1. Добавить элемент");

Console.SetCursorPosition(80, 2);

Console.WriteLine("2. Удалить элемент");

Console.SetCursorPosition(80, 3);

Console.WriteLine("3. Найти по слову");

Console.SetCursorPosition(80, 4);

Console.WriteLine("Ecs. Назад");

Console.ResetColor();

switch (Console.ReadKey().Key)

{

case ConsoleKey.D1:

Console.Clear();

OptopnAddM(ref data);

break;

case ConsoleKey.D2:

Console.Clear();

OptiondDeleteM(ref data);

break;

case ConsoleKey.D3:

Console.Clear();

OptionSearchM(ref data);

break;

case ConsoleKey.Escape:

exit = false;

break;

default:

break;

}

}while (exit);

}

static void OptionM()

{

Data data = new Data(99);

StreamReader stream = new StreamReader("Data.txt");

string str;

while ((str = stream.ReadLine()) != null)

{

Multiplications.Add(ref data, str);

}

stream.Close();

bool exit = true;

do

{

Console.Clear();

data.Print();

Console.ForegroundColor = ConsoleColor.Blue;

Console.SetCursorPosition(80, 0);

Console.WriteLine("Метод умножения");

Console.SetCursorPosition(80, 1);

Console.WriteLine("1. Добавить элемент");

Console.SetCursorPosition(80, 2);

Console.WriteLine("2. Удалить элемент");

Console.SetCursorPosition(80, 3);

Console.WriteLine("3. Найти по слову");

Console.SetCursorPosition(80, 4);

Console.WriteLine("Ecs. Назад");

Console.ResetColor();

switch (Console.ReadKey().Key)

{

case ConsoleKey.D1:

Console.Clear();

OptionAddD(ref data);

break;

case ConsoleKey.D2:

Console.Clear();

OptionDeleteD(ref data);

break;

case ConsoleKey.D3:

Console.Clear();

OptionSearchD(ref data);

break;

case ConsoleKey.Escape:

exit = false;

break;

default:

break;

}

}while(exit);

}

internal static void Main(string[] args)

{

Console.CursorVisible = false;

bool exit = true;

do

{

Console.WriteLine("1. Метод деления\n" +

"2. Метод умножения\n" +

"Esc. Завршить работу");

switch (Console.ReadKey().Key)

{

case ConsoleKey.D1:

Console.Clear();

OptionsD();

break;

case ConsoleKey.D2:

Console.Clear();

OptionM();

break;

case ConsoleKey.Escape:

exit = false;

break;

default:

break;

}

Console.Clear();

}while(exit);

}

}

}

namespace Hashing;

public class Game

{

public string Key { get; set; }

public int Id { get; set; }

public int Rating { get; set; }

public Game(string key, int id)

{

this.Key = key;

this.Id = id;

}

}

namespace Hashing;

public abstract class Divisions

{

private static int Function(string key, int m)

{

int k = 0;

foreach (var c in key)

{

k += c;

}

int hash = ((k % 163) + 1) % m;

return hash;

}

public static void Add(ref Data data, string key)

{

int h = Function(key, data.table.Count);

data.table[h].Add(new Game(key , h));

}

public static int Search(ref Data data, string key)

{

int h = Function(key, data.table.Count());

for (int i = 0; i < data.table[h].Count(); ++i)

{

if (data.table[h][i].Key == key)

{

return data.table[h][i].Id;

}

}

return -1;

}

public static bool CanDelete(ref Data table, string key)

{

int h = Function(key, table.table.Count());

for (int i = 0; i < table.table[h].Count(); ++i)

{

if (table.table[h][i].Key == key)

{

table.table[h].RemoveAt(i);

return true;

}

}

return false;

}

}

namespace Hashing;

public abstract class Multiplications

{

private static int Function(string key, int m)

{

int k = 0;

foreach (var c in key)

{

k += c;

}

double temp1 = (((Math.Sqrt(5) - 1) / 2) \* k) - Math.Truncate((((Math.Sqrt(5) - 1) / 2) \* k));

int hash = Convert.ToInt32(m \* temp1);

return hash;

}

public static void Add(ref Data data, string key)

{

int h = Function(key, data.table.Count);

data.table[h].Add(new Game(key , h));

}

public static int Search(ref Data data, string key)

{

int h = Function(key, data.table.Count());

for (int i = 0; i < data.table[h].Count(); ++i)

{

if (data.table[h][i].Key == key)

{

return data.table[h][i].Id;

}

}

return -1;

}

public static bool CanDelete(ref Data table, string key)

{

int h = Function(key, table.table.Count());

for (int i = 0; i < table.table[h].Count(); ++i)

{

if (table.table[h][i].Key == key)

{

table.table[h].RemoveAt(i);

return true;

}

}

return false;

}

}

namespace Hashing;

public class Data

{

public List<List<Game>> table = new List<List<Game>>();

public Data(int size)

{

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

{

table.Add(new List<Game>());

}

}

public void Print()

{

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

if (table[i].Count != 0)

{

int h = i;

string tmp = "";

for (int j = 0; j < table[h].Count; ++j)

{

if (table[h][j].Id == i)

{

tmp += table[h][j].Key + " -> ";

}

}

Console.WriteLine(i + ". " + tmp);

}

}

}