

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 «Хеширование»

**направление подготовки**

**09.03.03 Прикладная информатика**

**Прикладная информатика в экономике**

|  |  |
| --- | --- |
| Выполнил студент гр. Б9121-09.03.03 ПИЭ(2) | |
|  | Туровец В. Ю. |
|  | |
| Проверил ст. преподаватель | |
|  | Елсукова Е. А. |
|  | |
| не зачтено/зачтено | |

г. Владивосток

2022г.

Оглавление

[Условие задачи 3](#_Toc125459412)

[Функциональное описание 4](#_Toc125459413)

[Входные данные 4](#_Toc125459414)

[Структура программы 5](#_Toc125459415)

[Интерфейс программы 8](#_Toc125459416)

[Тестовые примеры 8](#_Toc125459417)

[Полученные результаты 11](#_Toc125459418)

[Приложение 12](#_Toc125459419)

Условие задачи

Изучение основных методов организации таблиц идентификаторов, получение представления о преимуществах и недостатках, присущих различным методам организации таблиц идентификаторов.

* Разработать **две** хэш-функции, предусмотреть обработку коллизий. Хеш–функции и метод для разрешения коллизий выбрать любые из указанных в теоретической части.
* *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.
* На основании полученных хэш-значений построить хэш-таблицу.
* Реализовать Поиск, Добавление, Удаление элементов.

ВХОДНЫЕ ДАННЫЕ: чтение данных осуществляется из файла.

Массив записей; (количество 20) структура:   
Название альбома, Исполнитель, Год выпуска (поле сортировки).

ВЫХОДНЫЕ ДАННЫЕ:

Таблица Вида Название альбома / Исполнитель / Год выпуска и хеши подвергшиеся коллизии (если имеются)

Функциональное описание

Входные данные



Рисунок 1 – Файл массива записей

public class Disk  
{  
 public string AlbumName { get; set; }  
 public string ArtistName { get; set; }  
 public int Year { get; set; }  
  
 public override string ToString() => $"{ArtistName} / {AlbumName} / {Year}";  
   
 public Disk(string albumName, string artistName, int year)  
 {  
 this.AlbumName = albumName;  
 this.ArtistName = artistName;  
 this.Year = year;  
 }  
  
 public Disk() { }  
   
}

Приложение 1 – Класс Disks

for (int i = 0; File.ReadLines(Path).ElementAtOrDefault(i) != null; i++) *//Из файла в hash*{  
 string? readText = File.ReadLines(Path).ElementAtOrDefault(i);  
 string[] text = (readText.Split( '\n').ToArray());  
  
 foreach (var part in text)  
 {  
 var temp = part.Split("; ");   
   
 string AlbumName, AutorName; int year;  
 AlbumName = temp[0]; AutorName = temp[1]; year = Int32.Parse(temp[2]);  
   
 hashTable.AddMiddle(new Disk(){AlbumName = AlbumName, ArtistName = AutorName , Year = year});  
   
 }  
}

Приложение 2 – Инициализация массива записей

Структура программы

Программа реализован на c#, обе хеш-функции выполнены в виде методов класса Program, которые вызываются методами предварительно превращающими все числа строки в число (поскольку хеш-функции работают с числами)

1. Хеш-функция колапса

public static long CollapseHash(Disk[] disk, long hashLenght)

1. Хеш-функция метода середины квадратов

internal static long MakeMiddleSuaresHash (Disk[] disk, long hashLenght)

1. Класс хеш-таблицы

public HashTable(int hashLength)

Метод «CollapseHash» получает на вход значения элемента класса Disk и длину хеша, после чего передаёт сумму всех char элементов строк превращённых в целые числа (int) и передаёт его одноимённому методу «CollapseHash», который складывает все цифры получая тем самым хеш.

public static long CollapseHash(Disk disk, long hashLenght)  
{  
 long data = 0;  
  
 foreach (var AlbumChar in disk.AlbumName) data += (int)AlbumChar;  
 foreach (var ArtistChar in disk.ArtistName) data += (int)ArtistChar;  
 data += disk.Year;  
  
 return CollapseHash(data, hashLenght);  
}

public static long CollapseHash(long data, long hashLength)  
{long hash = 0;  
  
 while (data != 0)  
 {  
 hash += data % hashLength;  
 data /= hashLength;  
 }  
  
 return hash % hashLength;  
  
}

Приложение 3 – Метод «Collapse Hash»

Метод «MakeMiddleSuaresHash» также получает на вход значения элемента класса Disk и длину хеша, после чего передаёт сумму всех char элементов строк превращённых в целые числа (int) и передаёт его методу «MiddleSuaresHash», который возводит число в квадрат и берёт из него середину, получая тем самым хеш.

public static long MakeMiddleSquaresHash(Disk disk, long hashLenght)  
{  
 long data = 0;  
  
 foreach (var AlbumChar in disk.AlbumName) data += (int)AlbumChar;  
 foreach (var ArtistChar in disk.ArtistName) data += (int)ArtistChar;  
  
 data += disk.Year;  
  
 return MiddleSquaresHash(data, hashLenght);  
}

public static long MiddleSquaresHash(long data, long hashLength)  
{  
 *//Возводит в квадрат и берёт единицу* data = data \* data;  
  
 int resLength = (int)Math.Log10(hashLength) + (hashLength % 10 != 0 ? 1 : 0 );  
 int dataLength = (int)Math.Log10(data) + (data % 10 != 0 ? 1 : 0 );  
 int shift = (dataLength - resLength) / 2;  
   
 string s\_hash = data.ToString();  
  
 s\_hash = s\_hash.Substring(shift , resLength);  
  
 return Convert.ToInt64(s\_hash);  
}

Приложение 4 – Метод «MakeMiddleSuaresHash»

Класс HashTable позволяет избежать коллизий записывая одинаковые элементы в таблицу (двумерный массив)

public HashTable(int hashLength)  
{  
 this.hashLength = hashLength;  
  
 table = new List<Disk>[hashLength];  
  
 for (int i = 0; i < hashLength; i++) table[i] = new List<Disk>();  
  
}

Приложение 5 – Класс HashTable

Интерфейс программы

Интерфейс состоит из двух уровней на первом можно выбрать тип хэша (см. рисунок 4), а на втором действие совершаемое с ним(см. рисунок 5).

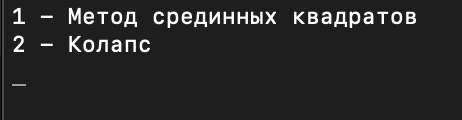


Рисунок 3 – Первый уровень интерфейса

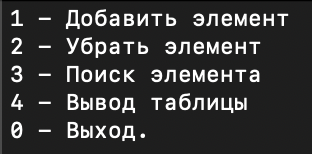


Рисунок 4 – Второй уровень интерфейса

Тестовые примеры

Приведены результаты функционала приложения.



Рисунок 5 – Результат метода коллапса

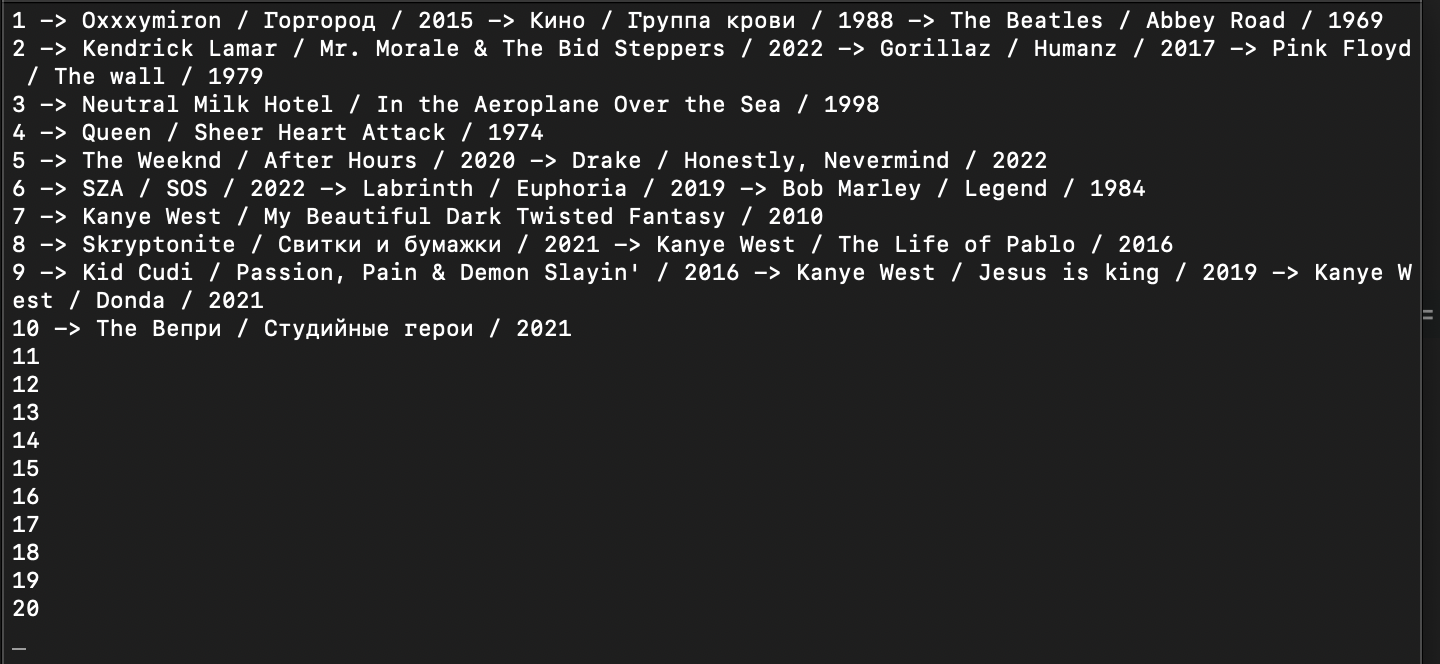


Рисунок 6 – Результат метода середины квадратов

Полученные результаты

Изучены основные методы организации таблиц идентификаторов. Метод коллапса имеет меньшее количество коллизий на исходных данных чем метод середины квадратов (результаты всегда будут зависеть от исходных данных)

Приложение

namespace Hash;  
  
public class Disk  
{  
 public string AlbumName { get; set; }  
 public string ArtistName { get; set; }  
 public int Year { get; set; }  
  
 public override string ToString() => $"{ArtistName} / {AlbumName} / {Year}";  
   
 public Disk(string albumName, string artistName, int year)  
 {  
 this.AlbumName = albumName;  
 this.ArtistName = artistName;  
 this.Year = year;  
 }  
  
 public Disk() { }  
   
}

Приложение 6 - Код класса Disk

namespace Hash  
{  
 internal class Program   
 {  
   
 public static long CollapseHash(long data, long hashLength)  
 {  
 *//Складывает все цыфры* long hash = 0;  
  
 while (data != 0)  
 {  
 hash += data % hashLength;  
 data /= hashLength;  
 }  
  
 return hash % hashLength;  
  
 }  
  
 public static long MiddleSquaresHash(long data, long hashLength)  
 {  
 *//Возводит в квадрат и берёт единицу* data = data \* data;  
  
 int resLength = (int)Math.Log10(hashLength) + (hashLength % 10 != 0 ? 1 : 0 );  
 int dataLength = (int)Math.Log10(data) + (data % 10 != 0 ? 1 : 0 );  
 int shift = (dataLength - resLength) / 2;  
   
 string s\_hash = data.ToString();  
  
 s\_hash = s\_hash.Substring(shift , resLength);  
  
 return Convert.ToInt64(s\_hash);  
 }  
  
   
 *//Два метода которые сумируют все символы, превращая Disk в число и после передают в методы,  
 //которые ищут хэш числа (с нагрузкой внутри)* public static long CollapseHash(Disk disk, long hashLenght)  
 {  
 long data = 0;  
  
 foreach (var AlbumChar in disk.AlbumName) data += (int)AlbumChar;  
 foreach (var ArtistChar in disk.ArtistName) data += (int)ArtistChar;  
 data += disk.Year;  
  
 return CollapseHash(data, hashLenght);  
 }  
   
 public static long MakeMiddleSquaresHash(Disk disk, long hashLenght)  
 {  
 long data = 0;  
  
 foreach (var AlbumChar in disk.AlbumName) data += (int)AlbumChar;  
 foreach (var ArtistChar in disk.ArtistName) data += (int)ArtistChar;  
  
 data += disk.Year;  
  
 return MiddleSquaresHash(data, hashLenght);  
 }   
  
  
 class HashTable  
 {  
  
 public HashTable(int hashLength)  
 {  
 this.hashLength = hashLength;  
  
 table = new List<Disk>[hashLength];  
  
 for (int i = 0; i < hashLength; i++) table[i] = new List<Disk>();  
  
 }  
  
  
 public void AddCollapse(Disk newValue)  
 {  
 var hash = CollapseHash(newValue , hashLength);  
  
 table[hash].Add(newValue);  
 }  
  
 public void AddMiddle(Disk newValue)  
 {  
 var hash = MakeMiddleSquaresHash(newValue , hashLength);  
  
 table[hash].Add(newValue);  
 }  
  
 public bool FindCollapse(Disk findElement)  
 {  
 var hash = CollapseHash(findElement , hashLength);  
  
 return table[hash].Contains(findElement);  
 }  
   
 public bool FindMidle(Disk findElement)  
 {  
 var hash = MakeMiddleSquaresHash(findElement , hashLength);  
  
 return table[hash].Contains(findElement);  
 }  
  
 public bool RemoveCollapse(Disk removeElement)  
 {  
 var hash = CollapseHash(removeElement , hashLength);  
  
 if (!table[hash].Contains(removeElement)) return false;  
  
 table[hash].Remove(removeElement);  
 return true;  
 }  
   
 public bool RemoveMidle(Disk removeElement)  
 {  
 var hash = MakeMiddleSquaresHash(removeElement , hashLength);  
  
 if (!table[hash].Contains(removeElement)) return false;  
  
 table[hash].Remove(removeElement);  
 return true;  
 }  
  
 public void Print()  
 {  
 for (int i = 0; i < hashLength; i++)  
 {  
 {Console.Write($"{i+1}");  
 foreach (var elem in table[i].ToArray())  
 {  
 if (elem != null) Console.Write($" -> {elem}");}  
 }  
 Console.WriteLine();  
 }  
 }  
  
 private List<Disk>[] table;  
 private int hashLength;  
 }  
  
 private static readonly string Path = "/Users/vlad/Desktop/Хэширование [Алг 3 лаб]/Third\_lab/Hash/Disks.txt";  
   
 public static void Main(string[] args)  
 {  
 *//Создание хэш-таблицы* HashTable hashTable = new HashTable(20);  
  
 Console.WriteLine("1 - Метод срединных квадратов");  
 Console.WriteLine("2 - Колапс");  
 int.TryParse(Console.ReadLine(), out var type);  
 if (type == 1)  
 {  
 var final = false;  
 do  
 {  
 *//Разделение из файла на отдельные элементы класа Disks* for (int i = 0; File.ReadLines(Path).ElementAtOrDefault(i) != null; i++) *//Из файла в hash* {  
 string? readText = File.ReadLines(Path).ElementAtOrDefault(i);  
 string[] text = (readText.Split( '\n').ToArray());  
   
 foreach (var part in text)  
 {  
 var temp = part.Split("; ");   
   
 string AlbumName, AutorName; int year;  
 AlbumName = temp[0]; AutorName = temp[1]; year = Int32.Parse(temp[2]);  
   
 hashTable.AddMiddle(new Disk(){AlbumName = AlbumName, ArtistName = AutorName , Year = year});  
   
 }  
 }  
 Console.Clear();  
 Console.WriteLine("1 - Добавить элемент");  
 Console.WriteLine("2 - Убрать элемент");  
 Console.WriteLine("3 - Поиск элемента");  
 Console.WriteLine("4 - Вывод таблицы");  
   
 Console.WriteLine("0 - Выход.");  
   
 string albumNameTemp, autorNameTemo; int yearTemp;  
 int.TryParse(Console.ReadLine(), out var switcher);  
 switch (switcher)  
 {  
 case 1:  
 Console.Clear();  
 var added = Console.ReadLine();  
 var temp = added.Split("; ");  
 albumNameTemp = temp[0]; autorNameTemo = temp[1]; yearTemp = Int32.Parse(temp[2]);  
 hashTable.AddMiddle(new Disk() {AlbumName = albumNameTemp, ArtistName = autorNameTemo , Year = yearTemp});  
 break;  
 case 2:  
 Console.Clear();  
 var removed = Console.ReadLine();  
 temp = removed.Split("; ");  
 albumNameTemp = temp[0];  
 autorNameTemo = temp[1];  
 yearTemp = Int32.Parse(temp[2]);  
 if (hashTable.RemoveMidle(new Disk()  
 { AlbumName = albumNameTemp, ArtistName = autorNameTemo, Year = yearTemp }))  
 {  
 Console.WriteLine("Элемент удалён");  
 }  
 else  
 {  
 Console.WriteLine("Такого элемента нет");  
 }  
   
 int.TryParse(Console.ReadLine(), out var waiter); *//ожидает пока пользователь нажмёт на enter* break;  
 case 3:  
 Console.Clear();  
 var finded = Console.ReadLine();  
 temp = finded.Split("; ");  
 albumNameTemp = temp[0];  
 autorNameTemo = temp[1];  
 yearTemp = Int32.Parse(temp[2]);  
 if ( hashTable.FindMidle(new Disk() { AlbumName = albumNameTemp, ArtistName = autorNameTemo, Year = yearTemp }) )  
 {  
 Console.WriteLine("Элемент присутствует");  
 }  
 else  
 {  
 Console.WriteLine("Элемент отсутствует");  
 }  
   
 int.TryParse(Console.ReadLine(), out waiter);  
 break;  
   
 case 4:  
 Console.Clear();  
 hashTable.Print();  
   
 int.TryParse(Console.ReadLine(), out waiter);  
 break;  
 case 0:  
 final = true;  
 break;  
 case >3:  
 break;  
 }  
   
 } while (!final);  
 }  
 else{var final = false;  
 do  
 {  
 *//Разделение из файла на отдельные элементы класа Disks* for (int i = 0; File.ReadLines(Path).ElementAtOrDefault(i) != null; i++) *//Из файла в hash* {  
 string? readText = File.ReadLines(Path).ElementAtOrDefault(i);  
 string[] text = (readText.Split( '\n').ToArray());  
   
 foreach (var part in text)  
 {  
 var temp = part.Split("; ");   
   
 string AlbumName, AutorName; int year;  
 AlbumName = temp[0]; AutorName = temp[1]; year = Int32.Parse(temp[2]);  
   
 hashTable.AddCollapse(new Disk(){AlbumName = AlbumName, ArtistName = AutorName , Year = year});  
   
 }  
 }  
 Console.Clear();  
 Console.WriteLine("1 - Добавить элемент");  
 Console.WriteLine("2 - Убрать элемент");  
 Console.WriteLine("3 - Поиск элемента");  
 Console.WriteLine("4 - Вывод таблицы");  
   
 Console.WriteLine("0 - Выход.");  
   
 string albumNameTemp, autorNameTemp; int yearTemp;  
 int.TryParse(Console.ReadLine(), out var switcher);  
 switch (switcher)  
 {  
 case 1:  
 Console.Clear();  
 var added = Console.ReadLine();  
 var temp = added.Split("; ");  
 albumNameTemp = temp[0]; autorNameTemp = temp[1]; yearTemp = Int32.Parse(temp[2]);  
 hashTable.AddCollapse(new Disk() {AlbumName = albumNameTemp, ArtistName = autorNameTemp , Year = yearTemp});  
 break;  
 case 2:  
 Console.Clear();  
 var removed = Console.ReadLine();  
 temp = removed.Split("; ");  
 albumNameTemp = temp[0];  
 autorNameTemp = temp[1];  
 yearTemp = Int32.Parse(temp[2]);  
 if(hashTable.RemoveCollapse(new Disk()  
 { AlbumName = albumNameTemp, ArtistName = autorNameTemp, Year = yearTemp }))  
 {  
 Console.WriteLine("Элемент удалён");  
 }  
 else  
 {  
 Console.WriteLine("Такого элемента нет");  
 }  
   
 int.TryParse(Console.ReadLine(), out var waiter); *//ожидает пока пользователь нажмёт на enter* break;  
 case 3:  
 Console.Clear();  
 var finded = Console.ReadLine();  
 temp = finded.Split("; ");  
 albumNameTemp = temp[0];  
 autorNameTemp = temp[1];  
 yearTemp = Int32.Parse(temp[2]);  
 if ( hashTable.FindCollapse(new Disk() { AlbumName = albumNameTemp, ArtistName = autorNameTemp, Year = yearTemp }) )  
 {  
 Console.WriteLine("Элемент присутствует");  
 }  
 else  
 {  
 Console.WriteLine("Элемент отсутствует");  
 }  
   
 int.TryParse(Console.ReadLine(), out waiter);  
 break;  
   
 case 4:  
 Console.Clear();  
 hashTable.Print();  
   
 int.TryParse(Console.ReadLine(), out waiter);  
 break;  
 case 0:  
 final = true;  
 break;  
 case >3:  
 break;  
 }  
   
 } while (!final);}  
   
   
 }  
 }  
}

Приложение 7 - Код основной программы