**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4

по дисциплине «Алгоритмы и структуры данных»

на тему: «Хэширование»

Выполнил: студент гр. ИП-31

Казутин П. Н.

Принял: ст. преподаватель

Косинов Г. П.

Гомель 2021

**Цель работы:** изучить основы построения хэш-функций и методами разрешения коллизий.

**Практическая часть:**

В задании необходимо обеспечить построение хэш-таблицы, а также добавление, удаление данных, поиск.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вариант | Вид хэширования | Вид хэширования | Метод разрешения коллизии | Доп. структуры |
| **9** | FNV | MurmurHash | Метод цепочек | Двунаправленный список |

***package by.ktsin.hashes;***

public class **MurMurHash** implements IHashProvider {

private static final int UNSIGNED\_MASK = 0xff;

private static final long UINT\_MASK = 0xFFFFFFFFL;

private static final long LONG\_MASK = 0xFFFFFFFFFFFFFFFFL;

private static final long m = 0xc6a4a793L;

private static final int r = 16;

@Override

public long hash(byte[] data, int seed) {

int length = data.length;

long h = seed ^ (length \* m);

// Mix 4 bytes at a time into the hash

int length4 = length >> 2;

for (int i = 0; i < length4; i++) {

final int i4 = i << 2;

long k = (data[i4] & UNSIGNED\_MASK);

k |= (data[i4 + 1] & UNSIGNED\_MASK) << 8;

k |= (data[i4 + 2] & UNSIGNED\_MASK) << 16;

k |= (long) (data[i4 + 3] & UNSIGNED\_MASK) << 24;

h = ((h + k) & UINT\_MASK);

h = ((h \* m) & UINT\_MASK);

h ^= ((h >> 16) & UINT\_MASK);

}

// remaining bytes

int offset = length4 << 2;

switch (length & 3) {

case 3:

h += ((data[offset + 2] << 16) & UINT\_MASK);

case 2:

h += ((data[offset + 1] << 8) & UINT\_MASK);

case 1:

h += ((data[offset]) & UINT\_MASK);

h = ((h \* m) & UINT\_MASK);

h ^= ((h >> r) & UINT\_MASK);

}

// final operations

h = ((h \* m) & UINT\_MASK);

h ^= ((h >> 10) & UINT\_MASK);

h = ((h \* m) & UINT\_MASK);

h ^= ((h >> 17) & UINT\_MASK);

// return the hash

return h >>> 1;

}

}

***package by.ktsin.hashes;***

public class **FNVHash** implements IHashProvider{

private static final long FNV\_64\_INIT = 0xcbf29ce484222325L;

private static final long FNV\_64\_PRIME = 1099511628211L;

@Override

public long hash(byte[] value, int seed) {

long rv = FNV\_64\_INIT;

final int len = value.length;

for (byte b : value) {

rv ^= b;

rv \*= FNV\_64\_PRIME;

}

return rv >>> 1;

}

}

***package by.ktsin.hashes;***

public interface **IHashProvider** {

public long hash(byte[] value, int seed);

}

***package by.ktsin;***

public class **KeyValuePair**<T> {

private final T key;

private final T value;

public KeyValuePair(T key, T value) {

this.key = key;

this.value = value;

}

public T getKey() {

return key;

}

public T getValue() {

return value;

}

@Override

public String toString(){

return String.format("[%s; %s]", key, value);

}

}

***package by.ktsin;***

import by.ktsin.hashes.IHashProvider;

import java.io.PrintStream;

import java.util.LinkedList;

public class **LabTable** {

private final IHashProvider hash;

private final int startSize;

private LinkedList[] array;

public LabTable(IHashProvider hash, int startSize) {

this.hash = hash;

this.startSize = startSize;

array = new LinkedList[startSize];

}

public void add(String key, String value){

int hval = (int) (hash.hash(key.getBytes(), 0) % startSize);

if(array[hval] != null){

array[hval].add(new KeyValuePair<String>(key, value));

}

else{

array[hval] = new LinkedList();

array[hval].add(new KeyValuePair<String>(key, value));

}

}

public void remove(String key){

int hval = (int) (hash.hash(key.getBytes(), 0) % startSize);

if(array[hval] != null){

if((long) array[hval].size() == 1){

array[hval] = null;

return;

}

else{

Object target = null;

for(Object i : array[hval]){

KeyValuePair<String> e = (KeyValuePair<String>) i;

if(e.getKey().equals(key)){

target = e;

}

}

if(target != null)

array[hval].remove(target);

}

}

else

return;

}

public KeyValuePair<String> search(String key){

int hval = (int) (hash.hash(key.getBytes(), 0) % startSize);

if(array[hval] != null){

if((long) array[hval].size() == 1){

return (KeyValuePair<String>)(array[hval].get(0));

}

else{

for(Object i : array[hval]){

KeyValuePair<String> e = (KeyValuePair<String>) i;

if(e.getKey().equals(key)){

return e;

}

}

return null;

}

}

else

return null;

}

public void print(PrintStream stream){

stream.printf("Hash table with size %d\n\n", startSize);

for(LinkedList list : array){

if(list != null){

for(Object obj : list){

stream.printf("%s, ", obj);

}

stream.println("");

}

}

}

}

***package by.ktsin;***

import by.ktsin.hashes.FNVHash;

import by.ktsin.hashes.MurMurHash;

import by.ktsin.hashes.IHashProvider;

import java.util.Random;

import java.util.Scanner;

public class **Main** {

public static void main(String[] args) {

FNVHash hash0 = new FNVHash();

MurMurHash hash1 = new MurMurHash();

Scanner sc = new Scanner(System.in);

LabTable table0 = fillTable(hash0, 100);

LabTable table1 = fillTable(hash1, 100);

System.out.println("First table:");

table0.print(System.out);

System.out.println("\n\n");

System.out.println("Second table:");

table1.print(System.out);

System.out.println("\n\n");

System.out.print("Enter new key: ");

String key = sc.nextLine();

System.out.print("Enter new value: ");

String value = sc.nextLine();

table0.add(key, value);

System.out.println("First table:");

table0.print(System.out);

System.out.println("\n\n");

table1.add(key, value);

System.out.println("Second table:");

table1.print(System.out);

System.out.println("\n\n");

System.out.print("Enter key to search: ");

key = sc.nextLine();

System.out.println("First table:");

System.out.println(table0.search(key));

System.out.println("\n\n");

System.out.println("Second table:");

System.out.println(table0.search(key));

System.out.println("\n\n");

System.out.print("Enter key to delete: ");

key = sc.nextLine();

table0.remove(key);

System.out.println("First table:");

table0.print(System.out);

System.out.println("\n\n");

table1.remove(key);

System.out.println("Second table:");

table1.print(System.out);

System.out.println("\n\n");

}

private static LabTable fillTable(IHashProvider provider, int count) {

LabTable table = new LabTable(provider, 997);

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

table.add(randString(), randString());

}

return table;

}

private static String randString() {

int leftLimit = 97; // letter 'a'

int rightLimit = 122; // letter 'z'

int targetStringLength = 5;

Random random = new Random();

return random.ints(leftLimit, rightLimit + 1)

.limit(targetStringLength)

.collect(StringBuilder::new, StringBuilder::appendCodePoint, StringBuilder::append)

.toString();

}

}

Результат выполнения:

First table:

Hash table with size 997

[nmlvh; joxvl],

[jkzmf; wkskp],

[dsrdo; gnuvx],

[ntnvj; ugkbq],

[fbftn; lbgkm],

Second table:

Hash table with size 997

[ycnsh; rifnq],

[tkcou; rhykw],

[cyxmm; yygnt],

[gkgbm; jlwyz],

[lhgmx; bfuxw],

Enter new key: test

Enter new value: borsch

First table:

Hash table with size 997

[test; borsch],

[nmlvh; joxvl],

[jkzmf; wkskp],

[dsrdo; gnuvx],

[ntnvj; ugkbq],

[fbftn; lbgkm],

Second table:

Hash table with size 997

[ycnsh; rifnq],

[tkcou; rhykw],

[test; borsch],

[cyxmm; yygnt],

[gkgbm; jlwyz],

[lhgmx; bfuxw],

Enter key to search: test

First table:

[test; borsch]

Second table:

[test; borsch]

Enter key to delete: test

First table:

Hash table with size 997

[nmlvh; joxvl],

[jkzmf; wkskp],

[dsrdo; gnuvx],

[ntnvj; ugkbq],

[fbftn; lbgkm],

Second table:

Hash table with size 997

[ycnsh; rifnq],

[tkcou; rhykw],

[cyxmm; yygnt],

[gkgbm; jlwyz],

[lhgmx; bfuxw],

**Вывод**: в ходе выполнения лабораторной работы изучил основы построения хэш-функций и методами разрешения коллизий.