Министерство науки и высшего образования Российской Федерации

Новосибирский Государственный технический университет

Кафедра автоматизированных систем управления



**Отчет по лабораторной работе №4**

**по дисциплине «Архитектура средств вычислительной техники»**

**«Хэш-таблицы»**

**Вариант–3**

Выполнили

студенты группы АВТ-813:

Кинчаров Данил

Пайхаев Алексей

Чернаков Кирилл

Преподаватель:

Ландовский Владимир Владимирович,

к.т.н., доцент кафедры АСУ

г. Новосибирск

2020 год

# **Цель работы:**

Изучить различные виды хеш-функций, познакомиться с хеш таблицами и приобрести навыки их программной реализации.

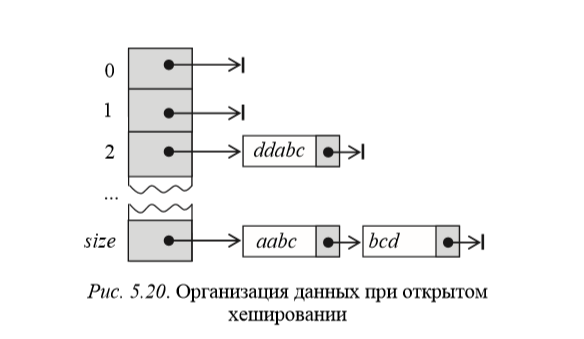
**Постановка задачи:**

Разработать программу, позволяющую добавлять элементы, подсчитывая число коллизий. Построить графики зависимости числа коллизий от текущего количества элементов. Провести эксперименты на одном множестве неповторяющихся элементов с использованием двух различных хеш-функций. Вариант 3, открытая хеш-таблица, тип элементов – Строка (наименование товара из прайс-листа).

**Схемы структур данных:**

Хеш-таблица – это структура данных, реализующая интерфейс ассоциативного массива и позволяющая хранение пар (ключ, значение), преимуществом которой является выполнение операции взятия нужного элемента за константное время. В хеш-таблице элементу отводится индекс h(k), где k – ключ элемента (в случае если элемент имеет сложную структуру, это может быть одно или несколько из его свойств), h – хеш-функция (функция, принимающая значения 0,…size-1, где size – размер хеш-таблицы).

Особенности открытого хеширования является хранение всех данных не в одном массиве, а помещение данных, имеющих одинаковое значение хеш-функции, в списки, указатели на которые и располагаются в массиве. Такой подход позволяет сильно упростить решение коллизий сравнительно с закрытым хешированием, а также разместить широкий диапазон возможных значений в малом количестве памяти.



**Описание выбранных хеш-функций:**

***Аддитивный метод для строк переменной длины:***

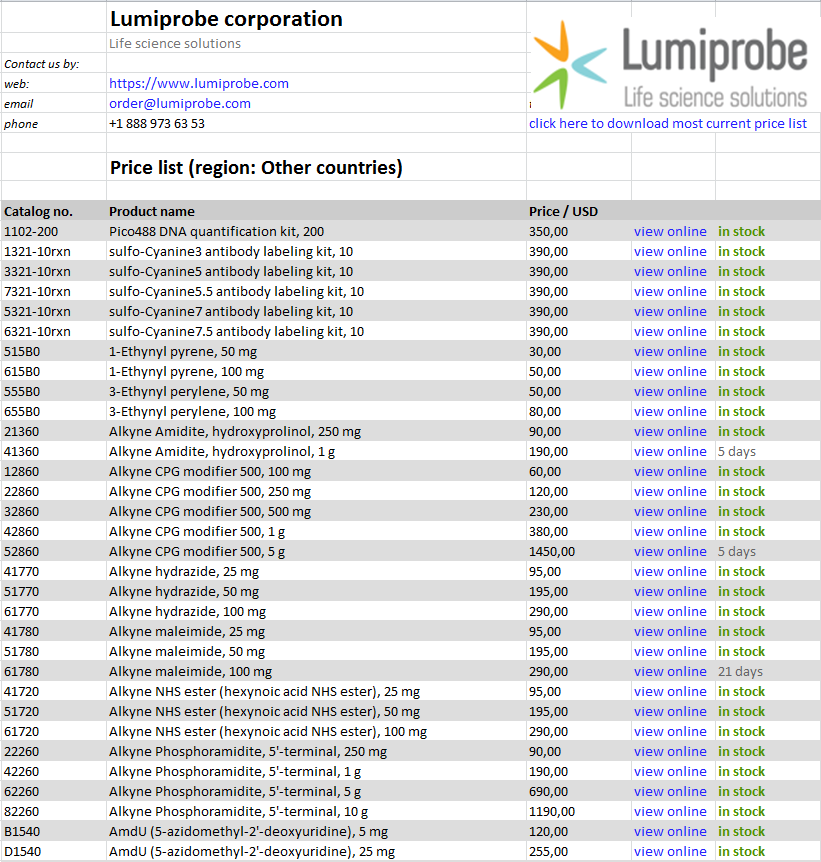
В данном методе идет сложение кодов символов, составляющих строку, после чего получившееся значение с остатком делится на размер таблицы. Рекомендуется использовать для строк, максимальный размер которых трудно оценить. Из недостатков можно отметить отсутствие учета позиции символа, что приводит к потенциальным коллизиям, если данный метод применять к разным строкам, содержащим одинаковый набор символов.

***Полиномиальный хэш:***

Начнем с определения. Пусть у нас есть строка s0..n-1. Полиномиальным хешем этой строки называется число h = hash(s0..n-1) = s0 + ps1 + p2s2 +… + pn-1sn-1, где p — некоторое натуральное число (чаще всего определяется длинной алфавита используемого для строк), а si — код i-ого символа строки s

**Результаты экспериментов в виде графиков:**

Для эксперимента был взят прайс-лист компании Lumiprobe corporation:

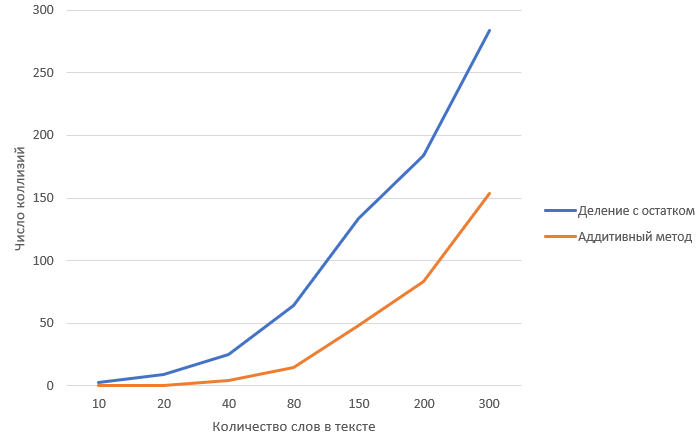


***Аддитивный метод для строк переменной длины:***

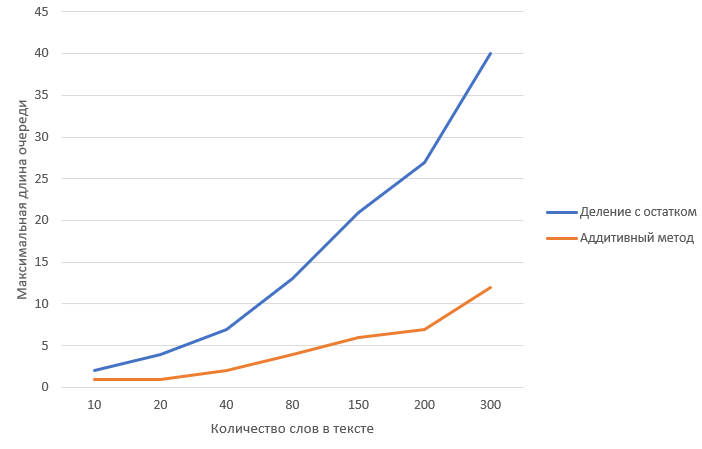
|  |  |
| --- | --- |
| Кол-во элементов | Число коллизий |
| 50 | 16 |
| 100 | 33 |
| 200 | 76 |
| 300 | 109 |
| 500 | 175 |
| 750 | 260 |
| 1000 | 354 |

***Полиномиальный хэш:***

Построим график зависимости числа коллизий от количества слов:



И график максимальной длины очереди в таблице. Нам это важно, так именно эта величина будет показывать сколько времени мы будем извлекать нужное нам значение из хэш-таблицы в худшем из случаев.



**Интерфейс:**

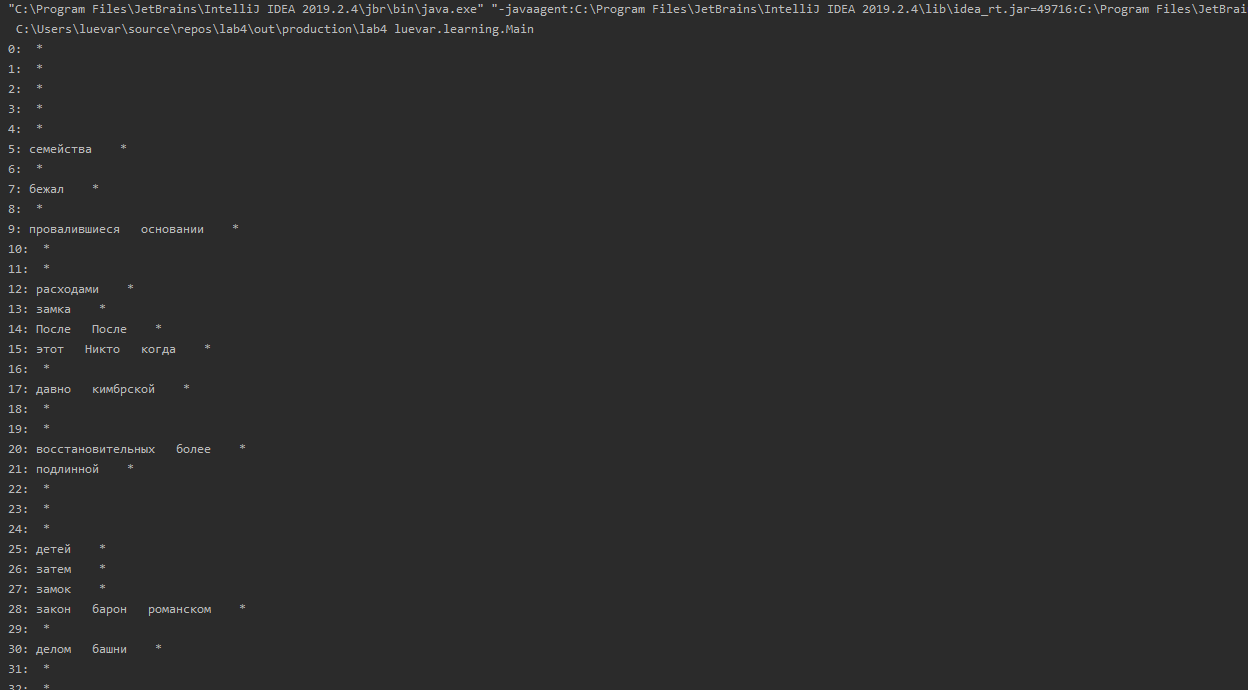


Рис. 8 Интерфейс

****

Рис. 9 Интерфейс

**Выводы:**

Таким образом, мы проверили, что способ хеширования деления строк с остатком во много раз уступает в равномерности распределения элементов в хэш-таблице аддитивному методу для строк переменной длины. Хоть график числа коллизий и в одном, и другом методе получился экспоненциальным, метод деления с остатком возрастает гораздо быстрее аддитивного. Время вычисления этих хэш функций аналогичен: O(n), где n-количество символов принятой строке. Большое количество коллизий ведет к увеличению максимального размера очереди в хэш таблице, что негативно влияет на общую эффективност, ибо время обращений к хэш-таблице для добавления или извлечения нового элемента будет обходиться не в константное время, а в O(k), где k-длина очереди, в которую мы пытаемся добавить/извлечь из нее элемент.

**Листинг программы:**

package luevar.learning.logic;  
  
import luevar.learning.service.HashTypes;  
  
import java.util.function.Function;  
  
public class HashMap {  
 private static final int *DEFAULT\_INITIAL\_CAPACITY* = 1 << 8;  
 private int capacity;  
 //Массив очередей  
 private Node[] table;  
 private Function<String, Integer> hash;  
 private Integer amountOfCollisions;  
 private Integer collisionDepth;  
  
 public HashMap(HashTypes hashType) {  
 capacity = *DEFAULT\_INITIAL\_CAPACITY*;  
 table = new Node[capacity];  
 amountOfCollisions = 0;  
 collisionDepth = 0;  
 switch (hashType) {  
 case *FIRST*: {  
 hash = s -> s.length() % capacity;  
 break;  
 }  
 case *SECOND*: {  
 hash = s -> {  
 int sum = 0;  
 for (int i = 0; i < s.length(); i++) {  
 sum += s.charAt(i);  
 }  
 return sum % capacity;  
 };  
 break;  
 }  
 case *THIRD*: {  
 hash = s -> {  
 int buff = Math.*abs*(s.hashCode());  
 return buff % capacity;  
 };  
 break;  
 }  
 }  
 }  
  
 static class Node {  
 final int key;  
 String value;  
 Node next;  
  
 Node(String value, int key, Node next) {  
 this.value = value;  
 this.key = key;  
 this.next = next;  
 }  
  
 public final int getKey() {  
 return key;  
 }  
  
 public final String getValue() {  
 return value;  
 }  
  
 public final String toString() {  
 return value + " ";  
 }  
  
 public final void setValue(String newValue) {  
 value = newValue;  
 }  
 }  
  
 private int hash(String value) {  
 return hash.apply(value);  
 }  
  
 private boolean tableContainsKey(String value) {  
 return table[hash(value)] != null;  
 }  
  
 //добавление элемента при коллизии  
 private void addNode(String value) {  
 amountOfCollisions++;  
 int localCollisionDepth = 1;  
 Node node = table[hash(value)];  
 while (node.next != null) {  
 node = node.next;  
 localCollisionDepth++;  
 }  
 node.next = new Node(value, hash(value), null);  
 localCollisionDepth++;  
 if (localCollisionDepth > collisionDepth) {  
 collisionDepth = localCollisionDepth;  
 }  
 }  
  
 //добавление элемента без коллизии  
 private void addQueue(String value) {  
 table[hash(value)] = new Node(value, hash(value), null);  
 if (collisionDepth == 0) {  
 collisionDepth = 1;  
 }  
 }  
  
 //метод добавления пары  
 public void assign(String value) {  
 try {  
 if (tableContainsKey(value)) {  
 addNode(value);  
 } else {  
 addQueue(value);  
 }  
 } catch (IndexOutOfBoundsException indexOutOfBounds) {  
 System.*out*.println("Index out of bounds!");  
 }  
 }  
  
 public StringBuilder toStringBuilder() {  
 StringBuilder builder = new StringBuilder();  
 Node node;  
 int i = 0;  
 for (Node value : table) {  
 node = value;  
 builder.append(i).append(": ");  
 while (node != null) {  
 builder.append(node.toString());  
 node = node.next;  
 }  
 builder.append(" \* ").append("\n");  
 i++;  
 }  
 return builder;  
 }  
  
 public String returnAmountOfCollisions() {  
 return "Amount of collisions: " + amountOfCollisions.toString() + "\n" + "Maximum size of single queue: " + collisionDepth.toString() + "\n";  
 }  
}

Класс, считывающий текст из файла, форматирующий его и записывающий отдельные слова в хэш-таблицу.

package luevar.learning.service;  
  
import luevar.learning.logic.HashMap;  
  
import java.io.\*;  
  
public class FileReading {  
 private final File inputFile;  
  
 public FileReading() {  
 String relativeDetailsFilePath = "./src/resources/lovecraft.txt";  
 inputFile = new File(relativeDetailsFilePath);  
 }  
  
 //считываем словарь в нашу хэш таблицу  
 public void readFromFile(HashMap hashMap) {  
 try (BufferedReader reader = new BufferedReader(new FileReader(inputFile))) {  
 String line;  
 String[] words;  
 while ((line = reader.readLine()) != null) {  
 words = line.replaceAll("[^а-яА-Яa-zA-Z ]", "").split("\\s+");  
 for (String word : words) {  
 if (!word.equals("")) {  
 hashMap.assign(word);  
 }  
 }  
 }  
 } catch (IOException io) {  
 System.*out*.println("Некорректная работа считывания файла.");  
 }  
 }  
}