ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | М. А. Мурашова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| «ХЕШИРОВАНИЕ ДАННЫХ» |
| по курсу: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4134к |  |  |  | Столяров Н.С. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

**1.1 Цель работы** Целью работы является изучение методов хеширования данных и получение практических навыков реализации хеш-таблиц.

**1.2 Задание на лабораторную работу** Составить хеш-функцию в соответствии с заданным вариантом и проанализировать ее. При необходимости доработать хеш-функцию. Используя полученную хеш-функцию разработать на языке программирования высокого уровня программу, которая должна выполнять следующие функции: − создавать хеш-таблицу; − добавлять элементы в хеш-таблицу; − просматривать хеш-таблицу; − искать элементы в хеш-таблице по номеру сегмента/по ключу; − выгружать содержимое хеш-таблицы в файл для построения гистограммы в MS Excel, или в аналогичном подходящем ПО; − удалять элементы из хеш-таблицы; − в программе должна быть реализована проверка формата вводимого ключа; − при удалении элементов из хэш-таблицы, в программе должен быть реализован алгоритм, позволяющий искать элементы, вызвавшие коллизию с удаленным; − в программе должен быть реализован алгоритм, обрабатывающий ситуации с переполнением хэш-таблицы.

**Вариант 17**



**Листинг**

**Проверка эффективности хеш функции**

Функция хеширования:

1. // хеширование
2. **int** My\_hash::hash(**char**\* key) {
3. **int** value = 1;
4. **for** (**int** i = 0; i < size\_key; i++) {
5. value += (**int**)key[i] \* (**int**)key[i];
6. }
7. **return** (value % count\_sigments);
8. }

Программа берет код каждого символа в строке и складывает их квадраты, деля на количество сегментов (В моем случае 2000)

**main.cpp**

#include <iostream>

**using** **namespace** std**;**

#include "libs/lib.h"

#include <cmath>

#include <time.h>

#include <iomanip>

// проверка ввода

#include "libs/simple\_char.h"

#include "libs/input\_validation.h"

//

#include "hash.h"

int menu**()** **{**

int id**;**

**while** **(true)** **{**

draw\_line**(**30**);**

cout **<<** "1) Сгенерировать ключ" **<<** endl**;**

cout **<<** "2) Ввести ключ вручную" **<<** endl**;**

cout **<<** "3) Сгенерировать список ключей" **<<** endl**;**

cout **<<** "4) Вывести список ключей" **<<** endl**;**

cout **<<** "5) Очистить список ключей" **<<** endl**;**

cout **<<** "6) Экспортировать в файл" **<<** endl **<<** endl**;**

cout **<<** "0) Выход" **<<** endl**;**

id **=** read\_value**(**" >>> "**,** **false,** **false,** **false);**

**if** **(**id **>=** 0 **&&** id **<=** 6**)** **{**

**return** id**;**

**}** **else** **{**

cout **<<** "Этого нет в меню" **<<** endl**;**

**}**

**}**

**}**

int main**()** **{**

// смена кодировки

system**(**"chcp 65001"**);**

srand**(**time**(NULL));**

/\*

i (int) = число

c (char) = буква

\*/

My\_hash my\_hash**(**"iiicii"**,** 6**);**

int menu\_i**;**

**while** **(true)** **{**

menu\_i **=** menu**();**

**switch** **(**menu\_i**)** **{**

**case** **(**0**):**

**return** 0**;**

**break;**

**case** **(**1**):**

my\_hash**.**generate**();**

**break;**

**case** **(**2**):**

my\_hash**.**generate**(false);**

**break;**

**case** **(**3**):**

int count**;**

**while** **(true)** **{**

count **=** read\_value**(**"Количество ключей: "**,** **false,** **false,** **false);**

**if** **(**count **>** 0**)** **{**

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

my\_hash**.**generate**();**

**break;**

**}** **else**

cout **<<** "Число должно быть больше 0." **<<** endl**;**

**}**

**break;**

**case** **(**4**):**

my\_hash**.**draw\_hash\_list**();**

**break;**

**case** **(**5**):**

my\_hash**.**clear\_hash\_list**();**

**break;**

**case** **(**6**):**

int file\_name\_length**;**

cout **<<** "Название файла: "**;**

char**\*** file\_name **=** get\_string**(&**file\_name\_length**);**

my\_hash**.**export\_to\_file**(**file\_name**);**

**break;**

**}**

**}**

cout **<<** my\_hash**.**generate**()** **<<** endl**;**

**return** 0**;**

**}**

**hash.cpp**

#include <iostream>

**using** **namespace** std**;**

#include <cstdio>

struct hash\_struct **{**

int hash**;**

char**\*** key**;**

**};**

class My\_hash **{**

public**:**

My\_hash**(**const char**\*** Key\_example**,** int Size\_key**,** int Count\_sigments**);**

**~**My\_hash**();**

int hash**(**char**\*);**

char**\*** generate**(**bool**);**

void draw\_hash\_list**();**

void clear\_hash\_list**();**

bool chek\_key**(**char**\*);**

bool find\_by\_hash**(**int**);**

bool find\_by\_key**(**char**\*);**

void export\_to\_file**(**char**\*);**

private**:**

void append\_list**(**char**\*);**

const char**\*** key\_example**;**

int size\_key**;**

int count\_sigments**;**

int size\_list **=** 0**;**

hash\_struct**\*** hash\_list**;**

**};**

// конструктор

My\_hash**::**My\_hash**(**const char**\*** Key\_example**,** int Size\_key**,** int Count\_sigments **=** 2000**)** **{**

key\_example **=** Key\_example**;**

size\_key **=** Size\_key**;**

hash\_list **=** **(**hash\_struct**\*)**malloc**(sizeof(**hash\_struct**));**

count\_sigments **=** Count\_sigments**;**

**}**

// деструктор

My\_hash**::~**My\_hash**()** **{**

free**(**hash\_list**);**

**}**

// хеширование

int My\_hash**::**hash**(**char**\*** key**)** **{**

int value **=** 1**;**

**for** **(**int i **=** 0**;** i **<** size\_key**;** i**++)** **{**

value **+=** **(**int**)**key**[**i**]** **\*** **(**int**)**key**[**i**];**

**}**

**return** **(**value **%** count\_sigments**);**

**}**

// генератор ключей

char**\*** My\_hash**::**generate**(**bool random **=** **true)** **{**

char**\*** key **=** **(**char**\*)** malloc**(**size\_key **\*** **sizeof(**char**));**

int length**;**

**while** **(true)** **{**

**if** **(**random**)** **{**

**for** **(**int i **=** 0**;** i **<** size\_key**;** i**++)** **{**

**if** **(**key\_example**[**i**]** **==** 'i'**)** key**[**i**]** **=** 48 **+** rand**()** **%** 10**;**

**if** **(**key\_example**[**i**]** **==** 'c'**)** key**[**i**]** **=** 65 **+** rand**()** **%** 26**;**

**}**

**}** **else** **{**

**while** **(true)** **{**

cout **<<** "Введите ключ формата " **<<** key\_example **<<** ": "**;**

key **=** get\_string**(&**length**);**

**if** **(**length **!=** size\_key **||** **!**chek\_key**(**key**))** **{**

cout **<<** "ключ не соответствует формату." **<<** endl**;**

**}** **else** **break;**

**}**

**}**

**if** **(!**find\_by\_key**(**key**))** **break;**

**}**

append\_list**(**key**);**

**return** key**;**

**}**

// проверка на правильность ключа (возвращает bool)

bool My\_hash**::**chek\_key**(**char**\*** key**)** **{**

**for** **(**int i **=** 0**;** i **<** size\_key**;** i**++)** **{**

**if** **(**key\_example**[**i**]** **==** 'i' **&&** **!(**key**[**i**]** **>=** 48 **&&** key**[**i**]** **<=** 57**))** **{return** **false;}**

**if** **(**key\_example**[**i**]** **==** 'c' **&&** **!(**toupper**(**key**[**i**])** **>=** 65 **&&** toupper**(**key**[**i**])** **<=** 90**))** **{return** **false;}**

**}**

**return** **true;**

**}**

// добавление ключа в список

void My\_hash**::**append\_list**(**char**\*** key**)** **{**

hash\_list **=** **(**hash\_struct**\*)**realloc**(**hash\_list**,** **(++**size\_list**)** **\*** **sizeof(**hash\_struct**));**

hash\_list**[**size\_list **-**1**].**key **=** **(**char**\*)** malloc**(**size\_key **\*** **sizeof(**char**));**

hash\_list**[**size\_list **-**1**].**key **=** key**;**

int hash\_bool **=** hash**(**key**);**

**while** **(true)** **{**

**if** **(!**find\_by\_hash**(**hash\_bool**))** **{**

hash\_list**[**size\_list **-**1**].**hash **=** hash\_bool**;**

**break;**

**}**

hash\_bool**++;**

**}**

**}**

// нахождение хеша в списке (возвращает bool)

bool My\_hash**::**find\_by\_hash**(**int hash**)** **{**

**for** **(**int i **=** 0**;** i **<** size\_list**;** i**++)** **{**

**if** **(**hash **==** hash\_list**[**i**].**hash**)** **{**

// cout << "1" << endl;

**return** **true;**

**}**

**}**

**return** **false;**

**}**

// нахождение ключа в списке (возвращает bool)

bool My\_hash**::**find\_by\_key**(**char**\*** key**)** **{**

**for** **(**int i **=** 0**;** i **<** size\_list**;** i**++)** **{**

**if** **(**key **==** hash\_list**[**i**].**key**)** **{**

// cout << "2" << endl;

**return** **true;**

**}**

**}**

**return** **false;**

**}**

// очистка списка

void My\_hash**::**clear\_hash\_list**()** **{**

hash\_list **=** **(**hash\_struct**\*)**malloc**(sizeof(**hash\_struct**));**

size\_list **=** 0**;**

**}**

// вывод ключей, хеша и id

void My\_hash**::**draw\_hash\_list**()** **{**

**for** **(**int i **=** 0**;** i **<** size\_list**;** i**++)** **{**

cout **<<** i **<<** " " **<<** hash\_list**[**i**].**hash **<<** " "**;**

draw\_char\_array**(**hash\_list**[**i**].**key**,** size\_key**);**

cout **<<** endl**;**

**}**

**}**

// экспорт списка в файл

void My\_hash**::**export\_to\_file**(**char**\*** file\_name**)** **{**

FILE **\***output\_file**;**

output\_file **=** fopen**(**file\_name**,** "w"**);**

**for** **(**int i **=** 0**;** i **<** size\_list**;** i**++)** **{**

fprintf**(**output\_file**,** "%d %d "**,** i**,** hash\_list**[**i**].**hash**);**

**for** **(**int j **=** 0**;** j **<** size\_key**;** j**++)**

fprintf**(**output\_file**,** "%c"**,** hash\_list**[**i**].**key**[**j**]);**

fprintf**(**output\_file**,** "\n"**);**

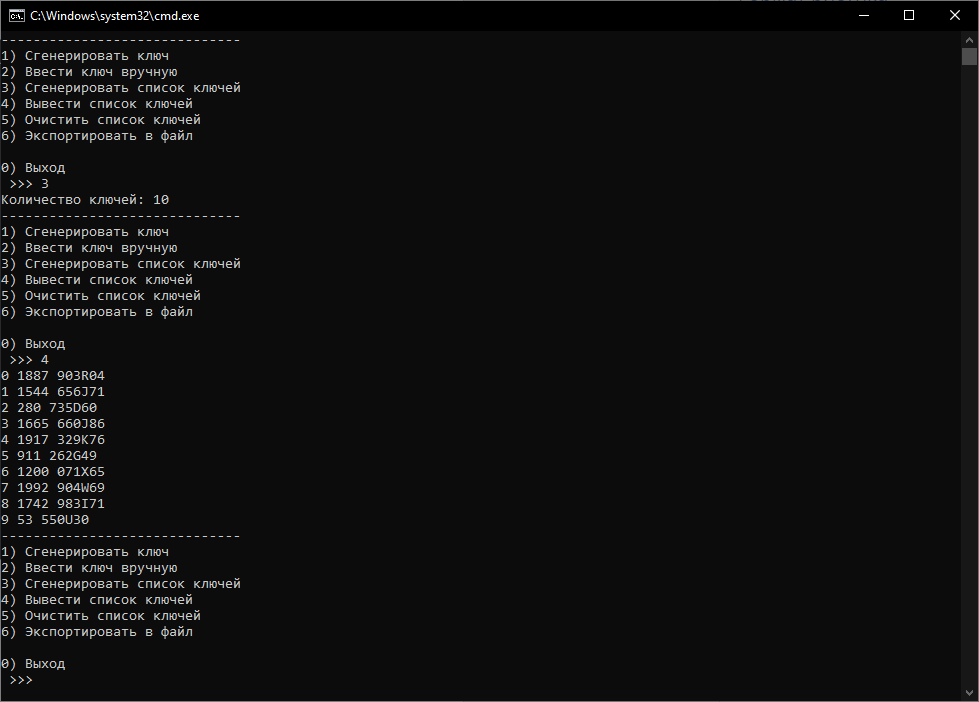
**}**

fclose**(**output\_file**);**

**}**

График идет на всей области определения равномерно.   
Диаграмма основана на 1000 ключей

**Результат работы**

****

**Вывод**

Я освоил методы ведения хештаблиц и хеширования.