ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | Рогачев С. А. |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4 |
| «ХЕШИРОВАНИЕ ДАННЫХ» |
| по курсу: Структуры и алгоритмы обработки данных |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | М011 |  |  |  | Борисов С. И. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2021

**Цель работы**

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

**Задание на лабораторную**

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.



**Листинг программы, реализующей алгоритм**

// М011 ЛБ-4 Боривов С. И. Вариант 10.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <fstream>

#include <ctime>

#include <cstring>

using namespace std;

const int n = 2000; //кол-во сегментов хеш-таблицы

const int m = 6; //длина ключа

const int m1 = 7; //длина ключа +1

const int tour = 14; //кол-во итераций поиска

const int c = 1; //шаг при разрешении коллизии

int hashi(char key[m1]) //возвращает номер сегмента

{

int hrez = 0, sum = 0;

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

sum += (int)((((int)key[i]) / (3 \* (i + 1)))) \* (int)key[i];

sum \*= sum \* 2;

sum = sum % 1000000;

hrez = (int)(sum / 100);

do

{

hrez = hrez % n;

} while (hrez > n - 1);

return hrez;

}

void hash\_table\_in\_file(char t[n][m1])

{

fstream file("hash.txt", ios\_base::out);

if (!file.is\_open())

{

cerr << "Не открыть файл\n";

return;

}

file.seekg(0);

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

{

{

file << i << ": ";

for (int j = 0; j < m; j++)

if (t[i][j] == '\0') break; else file << t[i][j];

file << '\n';

}

}

file.close();

}

void stinf(int t[n])

{

fstream file("stat.txt", ios\_base::out);

if (!file.is\_open())

{

cerr << "Не открыть файл\n";

return;

}

file.seekg(0);

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

{

file << t[i] << '\n';

}

file.close();

}

bool add(char t[n][m1], char k[m1])

{

int id;

id = hashi(k);

for (int i = 0; i < n \* tour; i++)

{

id = id + i \* c;

if (id >= n)

id = 0;//break;

if (!strcmp(t[id], "\0"))

{

strcpy(t[id], k);

return true;

}

}

return false;

}

short find(char t[n][m1], char k[m1])

{

int id;

for (unsigned long i = 0; i < n \* tour; i++)

{

id = hashi(k) + i \* c;

if (id >= n)

id = 0; //break;

if (!strcmp(t[id], "\0"))

return -1;

if (!strcmp(t[id], k))

return id;

}

return -1;

}

bool del(char t[n][m1], char k[m1])

{

if (find(t, k) == -1)

return false;

for (int i = 0; i < m1; i++)

{

strcpy(t[find(t, k)], "\0");

}

return true;

}

void print(char t[n][m1])

{

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

{

cout << i << ": ";

for (int j = 0; j < m1; j++)

{

if (t[i][0] == '\0') { cout << "\n"; break; }

else if (t[i][j] == '\0')

cout << "\n";

else

cout << t[i][j];

}

}

}

void find\_segment(char t[n][m1])

{

int index;

cout << "Введите номер сегмента: ";

cin >> index;

for (int i = 0; i < m1; i++)

{

cout << t[index][i];

}

cout << "\n";

}

int main()

{

setlocale(LC\_ALL, "rus");

srand(time(NULL));

int keys[n]; //массив результатов

char key[m1]; //ключ

int choice = -1;

char tab[n][m1];

char x[100];

while (choice != 0)

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "|1. Создать хеш-таблицу |\n";

cout << "|2. Заполнить полностью таблицу |\n";

cout << "|3. Добавить элементы в хеш-таблицу|\n";

cout << "|4. Просмотреть хеш-таблицу |\n";

cout << "|5. Поиск элемента в хеш-таблице |\n";

cout << "|6. Удалить элемент в хеш-таблице |\n";

cout << "|7. Сохранить хеш таблицу в файл |\n";

cout << "|8. Сохранить результаты в файл |\n";

cout << "|0. Выход |\n";

cout << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n";

cout << "\n";

do

{

cout << "Выберите пункт: ", cin >> x;

}

while (!isdigit(x[0]));

choice = atof(x);

cout << "Ожидание...\n";

cout << "\n";

switch (choice)

{

case 1:

{

//обнуление результирующего массива

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

{

for (int j = 0; j < m1; j++) tab[i][j] = '\0';

keys[i] = 0;

}

break;

}

case 2:

{

for (int i = 0; i < n \* 3; i++)

{

//генерация случайного ключа

for (int j = 0; j < m; j++)

{

if (j == 0 || j == 5 ) ///////////////////////////////

key[j] = 'A' + rand() % ('Z' - 'A');

else

key[j] = '0' + rand() % ('9' - '0');

}

key[m] = '\0';

//запись эксперементальных данных в массив keys

keys[hashi(key)]++;

if (find(tab, key) == -1)

add(tab, key);

} break;

}

case 3:

{

bool t;

int q = 1;

int r = 0;

do

{

cout << "Ввод ключа в формате БццццБ. Для выхода введите 0\n";

cout << " -> ", cin >> key;

if (key[0] == '0')

break;

for (int j = 0; j < m; j++)

{

if (j == 1 || j == 2 || j == 3 || j == 4)//////////

if (isdigit(key[j]))

{

t = true;

}

else

{

t = false;

break;

}

else if (j == 0 || j == 5 )//////////

if (isalpha(key[j]))

{

t = true;

}

else { t = false; break; }

}

if ((t == true) && (find(tab, key) == -1))

{

if (add(tab, key)) { cout << "Добавлено\n"; }

else { cout << "Переполнение!\n"; break; }

}

else

{

cout << "Ошибка введенного формата ключа\n"; break;

}

}

while (1);

break;

}

case 4:

{

print(tab);

break;

}

case 5:

{

int input;

cout << "1. Поиск по ключу\n2.Поиск по номеру сегмента\n";

cin >> input;

switch (input)

{

case 1: {cout << "Введите ключ: ", cin >> key;

if (find(tab, key) != -1)

{

cout << "Позиция: " << find(tab, key) << "\n";

}

else { cout << "Не найден\n"; }

break;

}

case 2: {find\_segment(tab); break; }

}

break;

}

case 6:

{

cout << "Введите ключ: ", cin >> key;

if (del(tab, key))

{

cout << "Удалено\n";

}

else

{

cout << "Ошибка\n";

}

break;

}

case 7:

{

hash\_table\_in\_file(tab);

break;

}

case 8:

{

stinf(keys);

break;

}

case 0:

{

return 0;

break;

}

default:

cout << "Ошибка\n";

}

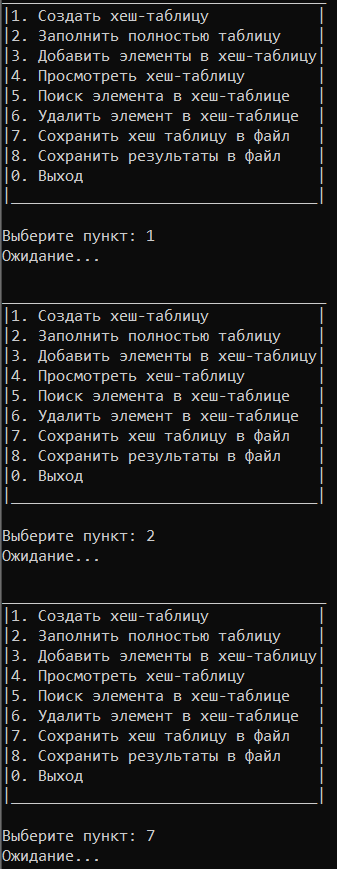
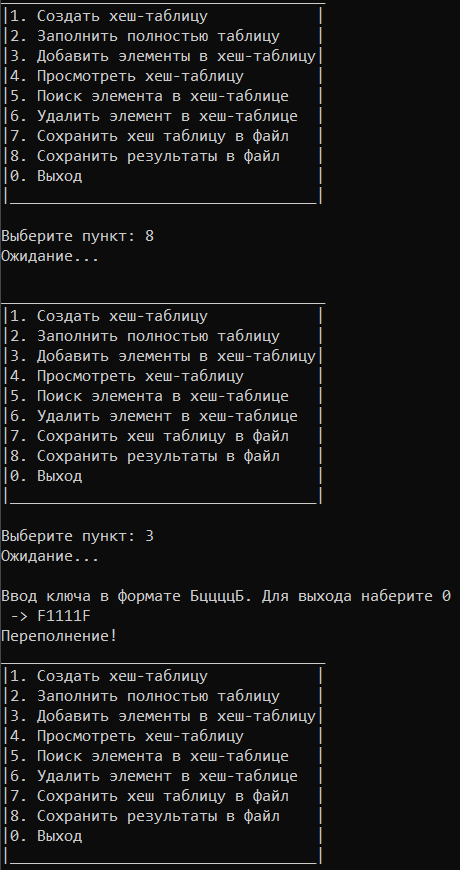
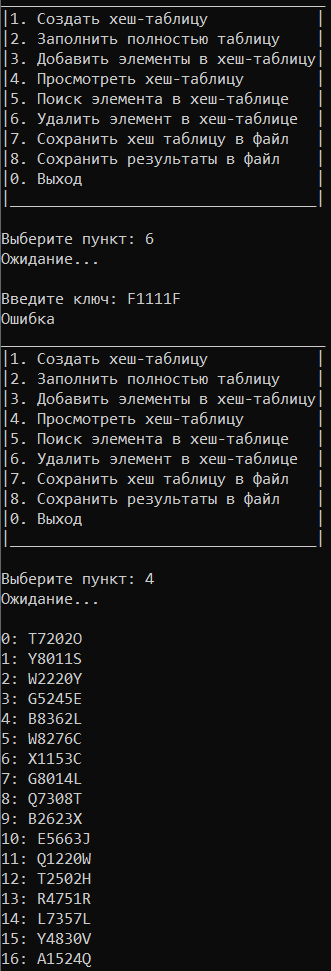
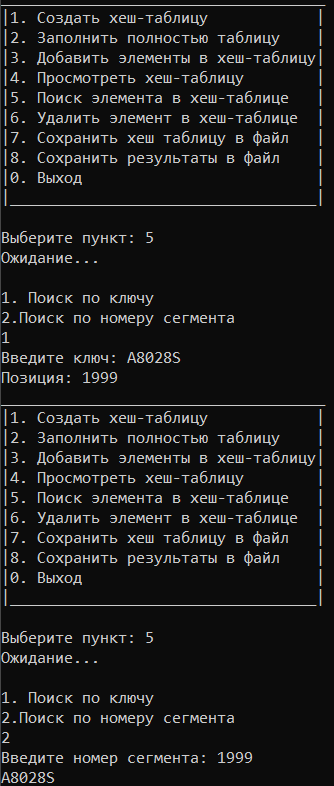
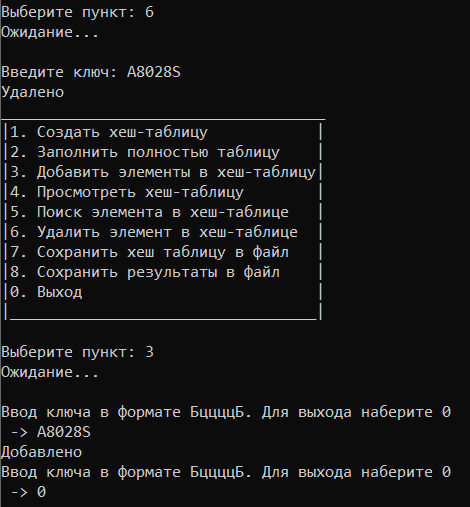
}

system("pause");

return 0;

}

**Примеры работы программы**

Будем вычислять адрес сегмента путем сложения возведенных в квадрат кодов символов ключа. Поскольку полученное при такой операции число превышает количество сегментов. Итоговую сумму квадратов возьмём по модулю от числа сегментов.



Анализ данной диаграммы показывает, что коллизии распределены равномерно, значит разработанную хеш-функцию можно считать приемлемой.

**Выводы**

На основе этих расчетов можно сделать вывод, что был разработан алгоритм, характеристики которого соответствуют поставленному заданию.