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 Иванов И.В.

подпись, дата инициалы, фамилия

Санкт-Петербург 2022

# Цель работы

Целью работы является изучение структур данных «стек» и «очередь», а также получение практических навыков их реализации.

# Вариант задания

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 хеш-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.



# Описание хеш-функций

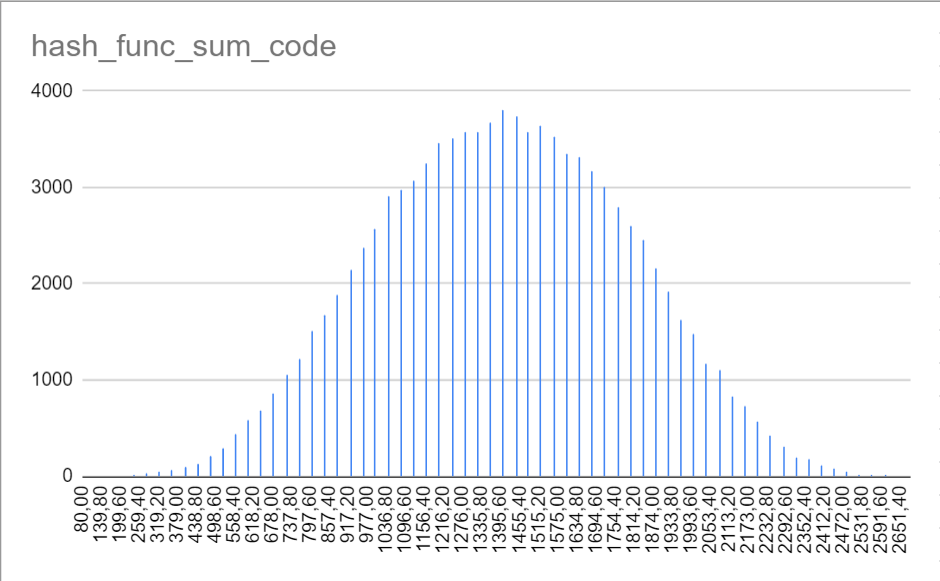
Хеш-функция int hash\_func\_sum\_code(string str). Функция суммирует коды символов из таблицы ASCII, вычитает минимально возможную сумму, умножает на 20 и берет остаток от количества сегментов. Распределяется неравномерно, большая часть концентрируется посередине.

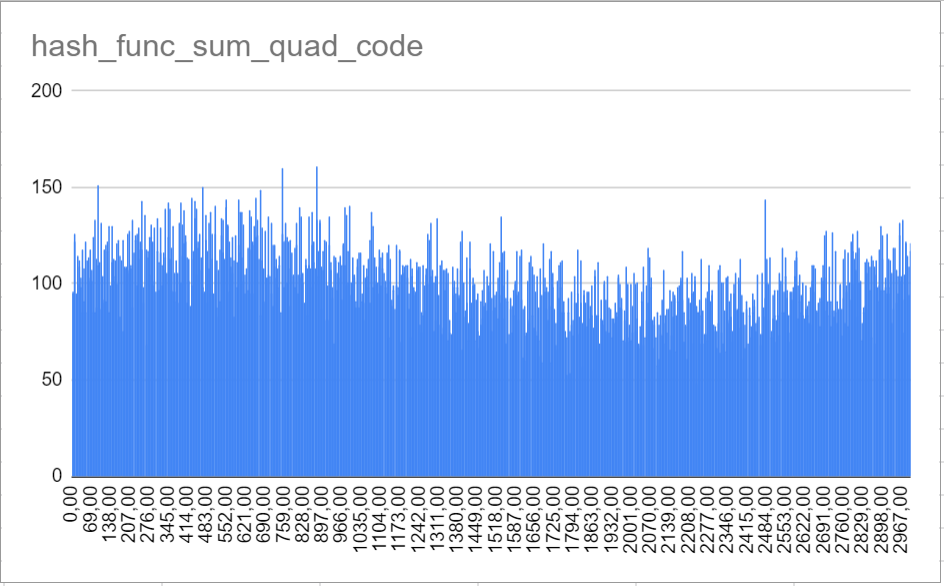
Хеш-функция int hash\_func\_sum\_quad\_code(string str). Функция суммирует квадраты кодов символов из таблицы ASCII и берет остаток от количества сегментов. Распределяет равномерно.

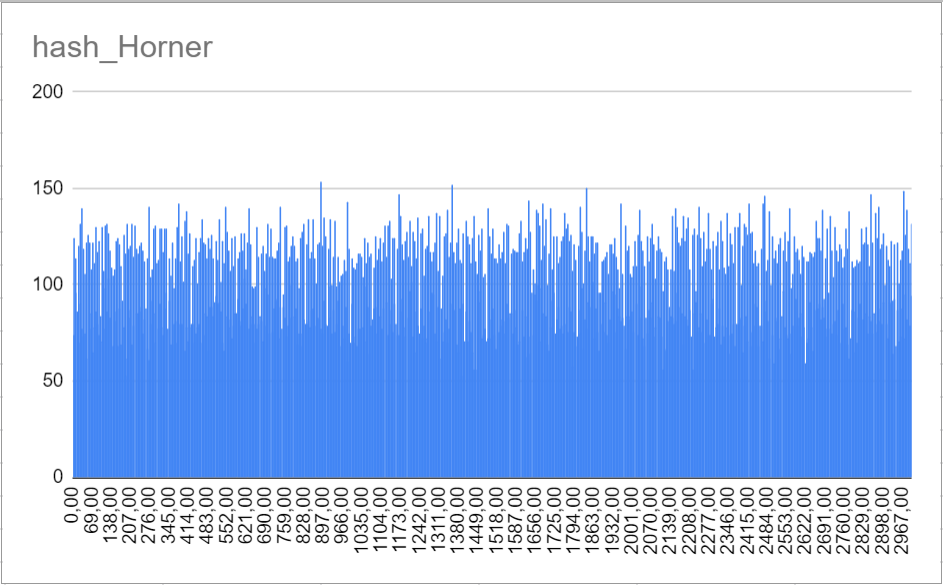
Хеш-функция int hash\_Horner(string str). Функция суммирует остатки от количества сегментов, полученные суммой кодов символов из таблицы ASCII и целочисленного деления количества сегментов на порядок символа в ключе. Далее полученный результат умножается на два, к нему прибавляется единица и находится остаток от деления на количество сегментов в хеш-таблице. Распределяет равномерно.

# Результаты анализа хеш-функций

Генерируем сто тысяч случайных ключей, обработаем их каждой хеш-функции, посчитаем количество коллизий, выгрузим полученные данные в excel и построим гистограммы.

Хеш-функция int hash\_func\_sum\_code(string str):

Хеш-функция int hash\_func\_sum\_quad\_code(string str):

Хеш-функция int hash\_Horner(string str):

Видно, что третья хеш-функция распределяет данные более равномерно, следовательно, возьмем ее для разработки хеш-таблицы.

**Листинг программы:**

Программа для оценки хеш-функций:

#include <iostream>

#include <ctime>

#include <fstream>

using namespace std;

class HashTable {

private:

constexpr static const double rehash\_size = 0.75;

struct Node {

bool deleted;

string data;

Node(const std::string& data) {

this->data = data;

deleted = false;

}

};

Node\*\* arr;

int size, size\_non\_nullptr, size\_buffer;

int hash\_Horner(std::string str) {

int hash\_result = 0;

for (int i = 0; i < 6; i++)

hash\_result += ((size\_buffer / (i + 10)) \* hash\_result + int(str[i])) % size\_buffer;

return (hash\_result \* 2) % size\_buffer;

}

public:

HashTable(int size) {

size\_buffer = size;

size\_non\_nullptr = this->size = 0;

arr = new Node \* [size\_buffer];

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

arr[i] = nullptr;

}

}

void Create(const int length) {

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

delete arr[i];

}

delete[] arr;

size\_buffer = length;

size\_non\_nullptr = this->size = 0;

arr = new Node \* [size\_buffer];

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

arr[i] = nullptr;

}

}

void Print() {

cout << "Хеш-таблица:" << endl;

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

if (arr[i] != nullptr && !arr[i]->deleted) {

cout << i + 1 << ": " << arr[i]->data << endl;

}

else {

cout << i + 1 << ":" << endl;

}

}

}

bool Find(const std::string data) {

int index = hash\_Horner(data);

int i = 0;

while (arr[index] != nullptr && i < size\_buffer) {

if (arr[index]->data == data && !arr[index]->deleted) {

std::cout << "Введенный ключ имеет индекс " << index << " в хеш-таблице." << std::endl;

return true;

}

index = (index + (i + 3 \* i \* i)) % size\_buffer;

++i;

}

return false;

}

void find\_el\_by\_ind() {

int ind;

cin >> ind;

if (arr[ind - 1] != nullptr && !arr[ind - 1]->deleted)

cout << "Ключ с индексом " << ind << "равен " << arr[ind - 1]->data << endl;

else

cout << " ячейка пуста " << endl;

}

bool Remove(const std::string data) {

int index = hash\_Horner(data);

int i = 0;

while (arr[index] != nullptr && i < size\_buffer) {

if (arr[index]->data == data && !arr[index]->deleted) {

arr[index]->deleted = true; // Cделать вывод элементов, вызвавших коллизию с удаляемым. Сделано!

--size;

i = 1;

index = (index + (2 \* i + 3 \* i \* i)) % size\_buffer;

while (arr[index] != nullptr) {

std::cout << "Элемент " << index << ": " << arr[index]->data << " вызывал коллизию с удаленным элементом" << std::endl;

++i;

index = (index + (2 \* i + 3 \* i \* i)) % size\_buffer;

}

return true;

}

index = (index + (2 \* i + 3 \* i \* i)) % size\_buffer;

++i;

}

return false;

}

void Resize() {

int old\_size\_buffer = size\_buffer;

size\_non\_nullptr = size = 0;

size\_buffer \*= 2;

Node\*\* arr2 = new Node \* [size\_buffer];

for (int i = 0; i < size\_buffer; ++i)

arr2[i] = nullptr;

swap(arr, arr2);

for (int i = 0; i < old\_size\_buffer; ++i) {

if (arr2[i] && !arr2[i]->deleted)

Add(arr2[i]->data); // добавляем элементы в новый массив

}

for (int i = 0; i < old\_size\_buffer; i++) {

delete arr2[i];

}

delete[] arr2;

}

void Rehash()

{

size\_non\_nullptr = 0;

size = 0;

Node\*\* arr2 = new Node \* [size\_buffer];

for (int i = 0; i < size\_buffer; ++i)

arr2[i] = nullptr;

std::swap(arr, arr2);

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

if (arr2[i] && !arr2[i]->deleted)

Add(arr2[i]->data);

}

for (int i = 0; i < size\_buffer; ++i)

if (arr2[i])

delete arr2[i];

delete[] arr2;

}

bool Add(const std::string data) {

if (size + 1 > int(rehash\_size \* size\_buffer))

Resize();

else if (size\_non\_nullptr > 2 \* size)

Rehash();

int index = hash\_Horner(data);

int i = 0, first\_deleted = -1;

while (arr[index] != nullptr && i < size\_buffer) {

if (arr[index]->data == data && !arr[index]->deleted)

return false;

if (arr[index]->deleted && first\_deleted == -1) // находим место для нового элемента

first\_deleted = index;

index = (index + (2 \* i + 3 \* i \* i)) % size\_buffer;

++i;

}

if (first\_deleted == -1) {

arr[index] = new Node(data);

++size\_non\_nullptr;

}

else {

arr[first\_deleted]->data = data;

arr[first\_deleted]->deleted = false;

}

++size;

return true;

}

~HashTable() {

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

delete arr[i];

}

delete[] arr;

}

};

void write(string file\_name, const int\* arr, const int length) {

std::fstream file;

file.open(file\_name, fstream::out);

if (file.is\_open())

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

file << arr[i] << '\n';

else

std::cerr << "ERROR!" << std::endl;

file.close();

}

bool check\_key(string key) {

bool check = true;

int length = 0;

for (int i = 0; i < 6; i++) {

if (i == 0 && (key[i] < 65 || key[i] > 90) || i == 4 && (key[i] < 65 || key[i] > 90) || i == 5 && (key[i] < 65 || key[i] > 90))

check = true;

else if (i == 1 && (key[i] < 48 || key[i] > 57) || i == 2 && (key[i] < 48 || key[i] > 57) || i == 3 && (key[i] < 48 || key[i] > 57))

check = true;

length++;

}

if (length != 6)

return false;

if (!check)

return false;

else

return true;

}

string create\_key() {

string key = "";

for (int i = 0; i < 6; i++) {

if (i == 0 || i == 4 || i == 5) {

key += char(rand() % 26 + 65);

}

else {

key += char(rand() % 10 + 48);

}

}

return key;

}

void menu() {

cout << "\n========MENU========" << endl;

cout << "0 - Создать таблицу(будет удалена предыдущая)" << endl;

cout << "1 - Сгенерировать" << endl;

cout << "2 - Добавить элемент" << endl;

cout << "3 - Вывести таблицу" << endl;

cout << "4 - Найти элемент" << endl;

cout << "5 - Удалить элемент" << endl;

cout << "6 - найти по индексу" << endl;

cout << "7 - Выход" << endl;

cout << "====================" << endl;

}

void select(HashTable& table) {

while (true) {

menu();

int choise;

cout << "Ваш выбор: ";

cin >> choise;

switch (choise) {

case 0: {

system("cls");

int size;

cout << "Введите размер таблицы: ";

cin >> size;

if (size > 3000)

cout << "Размер таблицы не должен превышать 3000" << endl;

else

table.Create(size);

break;

}

case 1: {

system("cls");

int number;

cout << "Введите количество сгенерируемых ключей: ";

cin >> number;

if (number > 3000)

cout << "Количество ключей не должно превышать 3000" << endl;

else {

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

table.Add(create\_key());

}

}

break;

}

case 2: {

system("cls");

string key;

cout << "Шаблон: БцццББ\nВведите ключ для добавления: ";

cin >> key;

if (check\_key(key))

table.Add(key);

else

cout << "Введенный ключ некорректен." << endl;

break;

}

case 3: {

system("cls");

table.Print();

break;

}

case 4: {

system("cls");

string key;

cout << "Шаблон: БцццББ\nВведите ключ для поиска: ";

cin >> key;

if (check\_key(key)) {

if (!table.Find(key))

cout << "Ключ " << key << " не найден!" << endl;

}

else

cout << "Введенный ключ некорректен." << endl;

break;

}

case 5: {

system("cls");

string key;

cout << "Шаблон: БцццББ\nВведите ключ для удаления и вывода коллизий: ";

cin >> key;

if (check\_key(key)) {

if (!table.Remove(key))

cout << "Ключ " << key << " не найден!" << endl;

}

else

cout << "Введенный ключ некорректен." << endl;

break;

}

case 6: {

table.find\_el\_by\_ind();

break;

}

case 7: {

exit(0);

break;

}

}

}

}

int main() {

srand(time(NULL));

setlocale(LC\_ALL, "RU");

HashTable Table(100);

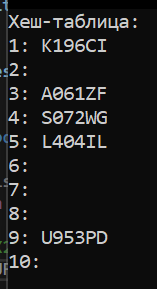
select(Table);

return 0;

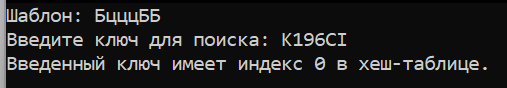
}

# Контрольные примеры

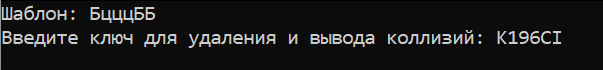
Изначально создается таблица с 3000 элементов, но по желанию можно ввести собственный размер таблицы, нажав ‘0’.



При нажатии на клавишу ‘1’ создается введенное количество ключей и распределяется по таблице. По клавише ‘3’ хеш-таблица выводится в консоль. По клавише ‘2’ возможно добавить ключ, введенный с клавиатуры, он должен соответствовать шаблону по варианту.

\

По клавише ‘4’ происходит поиск введенного с клавиатуры ключа. Если ключ отсутствует в таблице, то будет выведено соответствующее сообщение.



По клавише ‘5’ происходит удаление введенного с клавиатуры ключа и вывод ключей, вызывавших коллизию с удаленным. Если ключ отсутствует в таблице, то будет выведено соответствующее сообщение.

# Вывод

Изучены методы хеширования данных и получены практических навыков реализации хеш-таблиц. Выполнено задание в соответствии с вариантом №3.