МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФБГОУ ВО «ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра «Программная инженерия»

**Отчёт**

По лабораторной работе №4

**«Оценка качества хеш-функций»**

**Выполнили:**

Студенты группы 92-ПГ

Погосян Ж.В.

**Проверили:**

Ужаринский А.Ю.

Орёл — 2021

**Код программы:**

#include <chrono>

#include <random>

#include <cmath>

#include <map>

#include <iostream>

#include <fstream>

using namespace std;

const int r[6] = { 257, 269, 277, 389, 359, 491 };

//аддитивный метод деления ключа

map<int, int> adddev\_hash(vector<string>& keys, int M) {

int sum = 0;

map<int, int> result;

int key;

for (auto& str : keys) {

sum = 0;

for (auto x : str) {

sum += static\_cast<int>(x);

}

key = sum % M;

if (result.find(key) == result.end())

result[key] = 0;

else

result[key]++;

}

return result;

}

//аддитивный метод умножения ключа

map<int, int> addmult\_hash(vector<string>& keys, int M) {

int sum = 0;

const double A = (sqrt(5) - 1) / 2;

map<int, int> result;

int key;

for (auto& str : keys) {

sum = 0;

for (auto x : str) {

sum += static\_cast<int>(x);

}

key = static\_cast<int>(M \* ((sum \* A) - static\_cast<int>(sum \* A)));

if (result.find(key) == result.end())

result[key] = 0;

else

result[key]++;

}

return result;

}

//метод исключающего "или" и метод деления ключа

map<int, int> xordev\_hash(vector<string>& keys, int M) {

int sum = 0;

map<int, int> result;

int key;

for (auto& str : keys) {

sum = 0;

int counter = 0;

for (auto x : str)

sum += static\_cast<int>(x) ^ r[counter++];

key = sum % M;

if (result.find(key) == result.end())

result[key] = 0;

else

result[key]++;

}

return result;

}

//метод исключающего или и метод умножения ключа

map<int, int> xormult\_hash(vector<string>& keys, int M) {

int sum = 0;

const double A = (sqrt(5) - 1) / 2;

map<int, int> result;

int key;

for (auto& str : keys) {

sum = 0;

int counter = 0;

for (auto x : str)

sum += static\_cast<int>(x) ^ r[counter++];

key = static\_cast<int>(M \* ((sum \* A) - static\_cast<int>(sum \* A)));

if (result.find(key) == result.end())

result[key] = 0;

else

result[key]++;

}

return result;

}

void generate\_keys(int num, vector<string>& result) {

result.clear();

chrono::system\_clock::time\_point seed = chrono::system\_clock::now();

default\_random\_engine engine(seed.time\_since\_epoch().count());

uniform\_int\_distribution<int> distrib(48, 122);

int random;

result.resize(num);

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

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

random = distrib(engine);

if ((random >= 48 && random <= 57) || (random >= 65 && random <= 90) || (random >= 97 && random <= 122))

result.at(i).push\_back(static\_cast<char>(random));

else

--k;

}

}

}

int main(int argc, char\*\* argv) {

setlocale(LC\_ALL, "Russian");

if (argc < 3) {

cout << "Введите количество ключей и размер таблицы!" << endl; //

return 1;

}

int keys\_num = atoi(argv[1]);

int table\_size = atoi(argv[2]);

vector<string> keys;

generate\_keys(keys\_num, keys);

map<int, int> adddev = adddev\_hash(keys, table\_size);

map<int, int> xordev = xordev\_hash(keys, table\_size);

map<int, int> addmult = addmult\_hash(keys, table\_size);

map<int, int> xormult = xormult\_hash(keys, table\_size);

char input;

while (true) {

cout << "1. Сгенерировать коллизии" << endl;

cout << "2. Общее число коллизий для каждого метода" << endl;

cout << "3. Cреднее число коллизий на основании нескольких опытов" << endl;

cout << "4. Сгенерировать новые ключи" << endl;

cout << "5. Выйти" << endl;

input = getchar();

switch (input) {

case '1': {

ofstream os("colfile", ios::out | ios::trunc);

if (!os.is\_open()) {

cout << "Невозможно открыть файл отчёта";

break;

}

os << "Additive Divide" << endl;

for (auto& x : adddev)

os << x.second << endl;

os << "Additive Multiply" << endl;

for (auto& x : addmult)

os << x.second << endl;

os << "XOR Divide" << endl;

for (auto& x : xordev)

os << x.second << endl;

os << "XOR Multiply" << endl;

for (auto& x : xormult)

os << x.second << endl;

os.close();

cout << "Файл успешно сформирован!" << endl;

input = getchar();

break;

}

case '2': {

int collisions = 0;

for (auto& x : adddev)

collisions += x.second;

cout << "Аддитивный метод деления имеет " << collisions << " коллизий" << endl;

collisions = 0;

for (auto& x : addmult)

collisions += x.second;

cout << "Аддитивный метод умножения имеет " << collisions << " коллизий" << endl;

collisions = 0;

for (auto& x : xordev)

collisions += x.second;

cout << "Xor метод деления имеет " << collisions << " коллизий" << endl;

collisions = 0;

for (auto& x : xormult)

collisions += x.second;

cout << "Xor метод умножения имеет " << collisions << " коллизий" << endl;

input = getchar();

break;

}

case '3': {

int experiments = 1000;

double adddev\_collisions = 0;

double addmult\_collisions = 0;

double xordev\_collisions = 0;

double xormult\_collisions = 0;

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

generate\_keys(keys\_num, keys);

adddev = adddev\_hash(keys, table\_size);

addmult = addmult\_hash(keys, table\_size);

xordev = xordev\_hash(keys, table\_size);

xormult = xormult\_hash(keys, table\_size);

for (auto& x : adddev)

adddev\_collisions += x.second;

for (auto& x : addmult)

addmult\_collisions += x.second;

for (auto& x : xordev)

xordev\_collisions += x.second;

for (auto& x : xormult)

xormult\_collisions += x.second;

}

cout << "Среднее количество коллизий:" << endl;

cout << "аддитивный метод деления = " << adddev\_collisions / experiments << endl;

cout << "аддитивный метод умножения = " << addmult\_collisions / experiments << endl;

cout << "XOR метод деления = " << xordev\_collisions / experiments << endl;

cout << "XOR метод умножения = " << xormult\_collisions / experiments << endl;

input = getchar();

break;

}

case '4': {

generate\_keys(keys\_num, keys);

adddev = adddev\_hash(keys, table\_size);

addmult = addmult\_hash(keys, table\_size);

xordev = xordev\_hash(keys, table\_size);

xormult = xormult\_hash(keys, table\_size);

cout << "Сгенерированно " << keys\_num << " новых ключей" << endl;

input = getchar();

break;

}

case '5': {

return 0;

}

}

}

}

**Коллизии:**

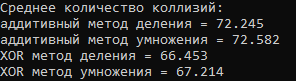


Рисунок 1 - среднее количество коллизий при 200 ключах

и размере таблицы 400

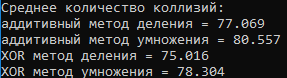


Рисунок 2 - среднее количество коллизий при 200 ключах

и размере таблицы 200

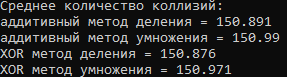


Рисунок 3 - среднее количество коллизий при 200 ключах

и размере таблицы 50

**Графики рассеивания ключей:**

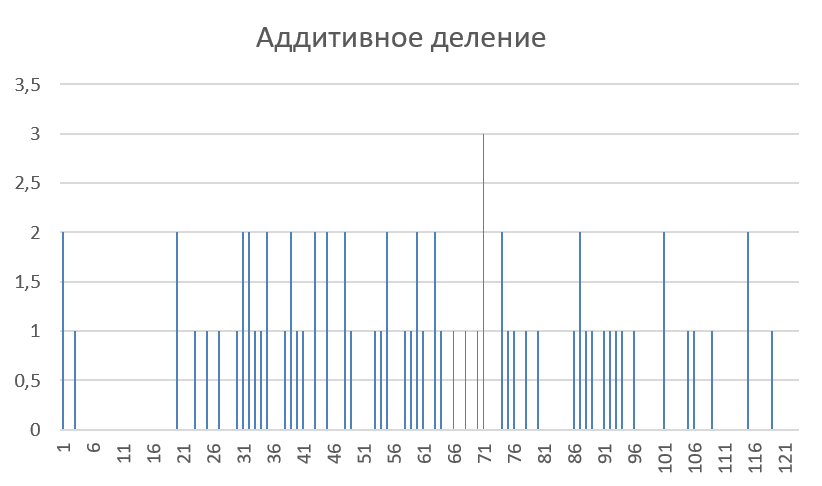


Рисунок 4 – использование аддитивного метода деления

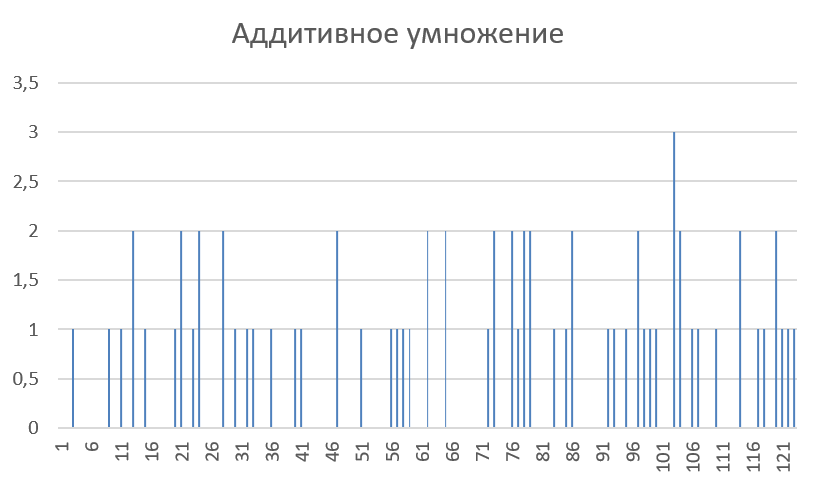


Рисунок 5 - использование аддитивного метода умножения

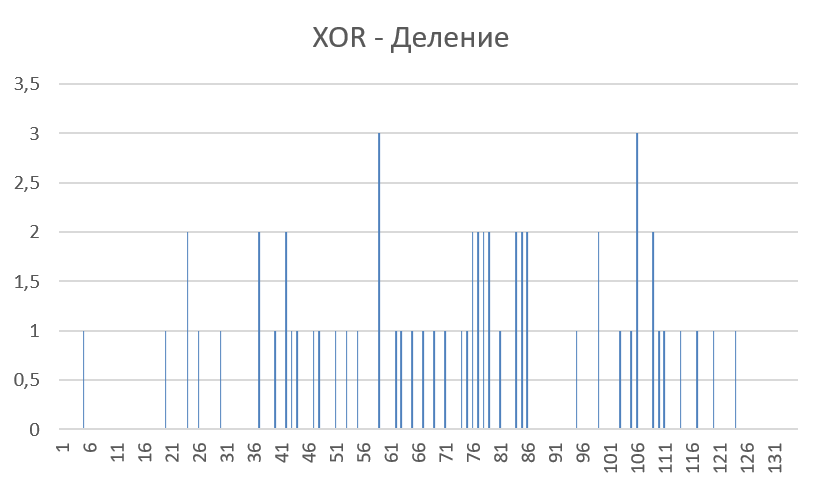


Рисунок 6 - использование метода исключающего или деления

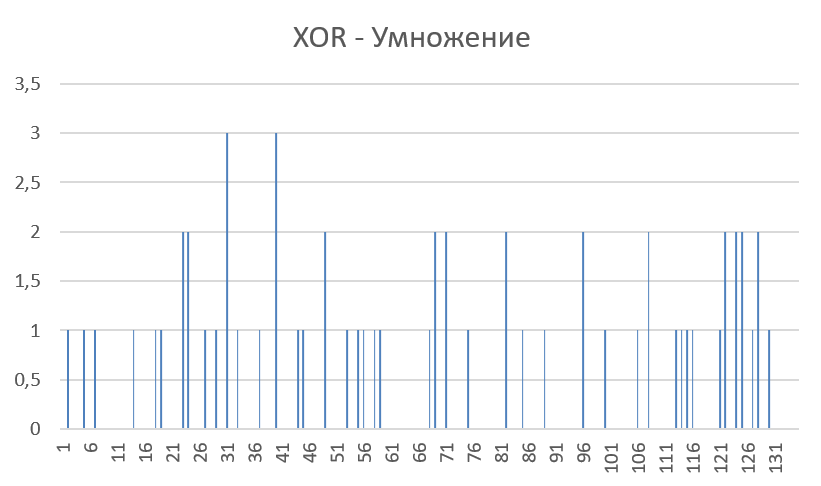


Рисунок 7 - использование метода исключающего или умножения

**Вывод:**

На основании результатов измерения рассеивания ключей делаем вывод о том, что использование метода исключающего или даёт более равномерное рассеивание коллизий. Соответственно, использование метода исключающего или считаем более предпочтительным.

Использование метода исключающего или помогает уменьшить количество коллизий тогда, когда количество ключей достаточно велико и размер хеш-таблицы больше или равен количеству ключей. В остальных случаях результаты схожи с аддитивным.

Измерения показали, что метод деления показывает себя несколько лучше метода умножения, что, возможно, связано с неподходящим выбором значения для константы A. При большом размере хеш-таблицы оба метода дают идентичное количество коллизий.