ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ­­

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

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4 |
| ХЕШИРОВАНИЕ ДАННЫХ |
| по дисциплине: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4631 |  |  |  | С.А. Гришин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2017

**1. Цель работы**

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

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:

1. создавать хеш-таблицу;
2. добавлять элементы в хеш-таблицу;
3. просматривать хеш-таблицу;
4. искать элементы в хеш-таблице по номеру сегмента/по ключу;
5. выгружать содержимое хеш-таблицы в файл для построения гистограммы в MS Excel, или в аналогичном подходящем ПО;
6. удалять элементы из хеш-таблицы;
7. в программе должна быть реализована проверка формата вводимого ключа;
8. при удалении элементов из хэш-таблицы, в программе должен быть реализован алгоритм, позволяющий искать элементы, вызвавшие коллизию с удаленным;
9. в программе должен быть реализован алгоритм, обрабатывающий ситуации с переполнением хэш-таблицы.

|  |  |  |  |
| --- | --- | --- | --- |
| **№ вар.** | **Формат ключа** | **Количество сегментов** | **Метод хеширования (разрешения коллизий)** |
| 12 | БццццБ | 2500 | Двойное хеширование |

**2. Описание хеш-функции**

Для первой хеш-функции использовался полиномиальный хеш;

*h1 = hash(s0..n-1) = s0 + ps1 + p2s2 +… + pn-1sn-1*, где *p* — некоторое натуральное число , а *si* — код *i*-ого символа строки *s*.

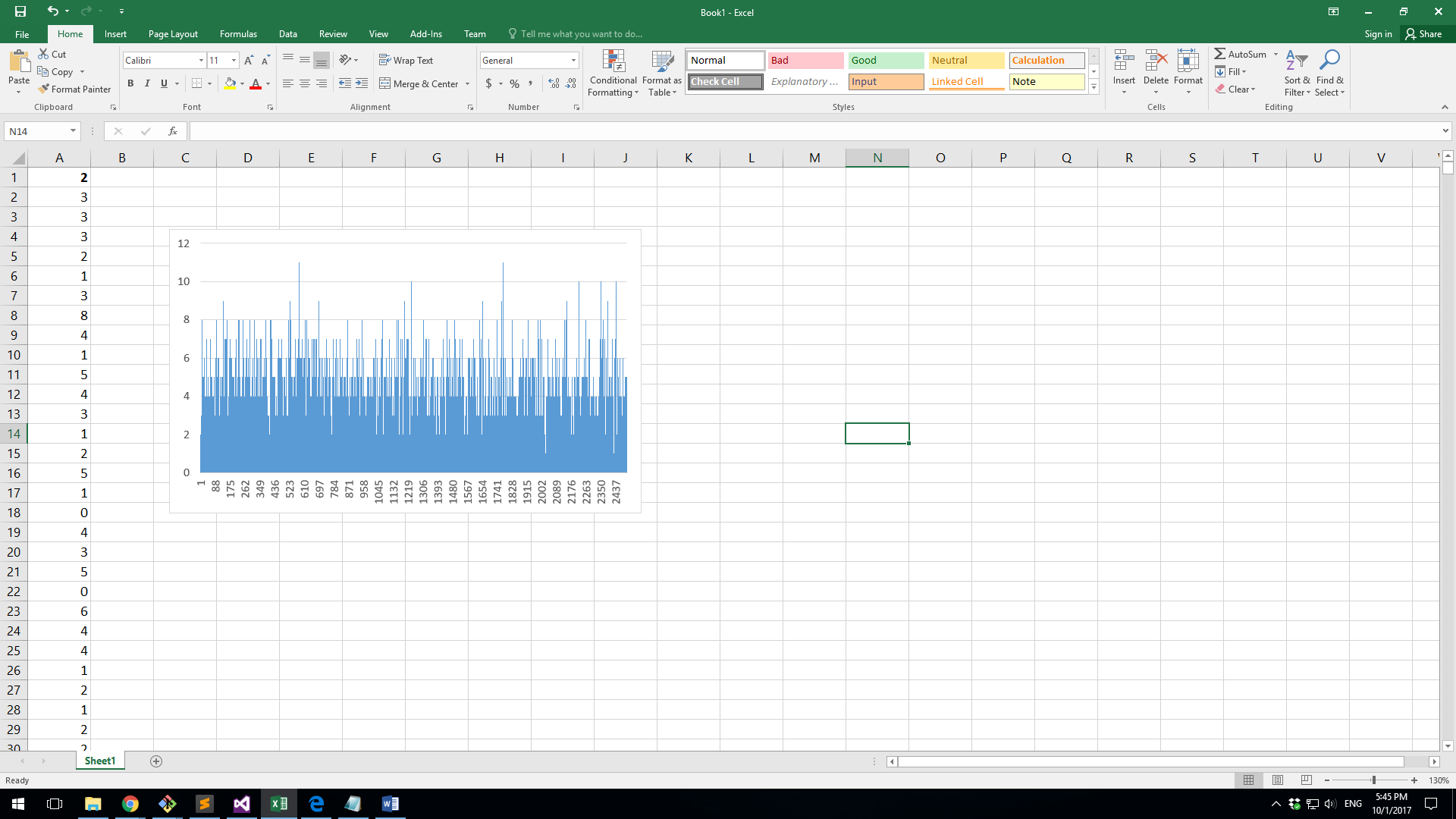
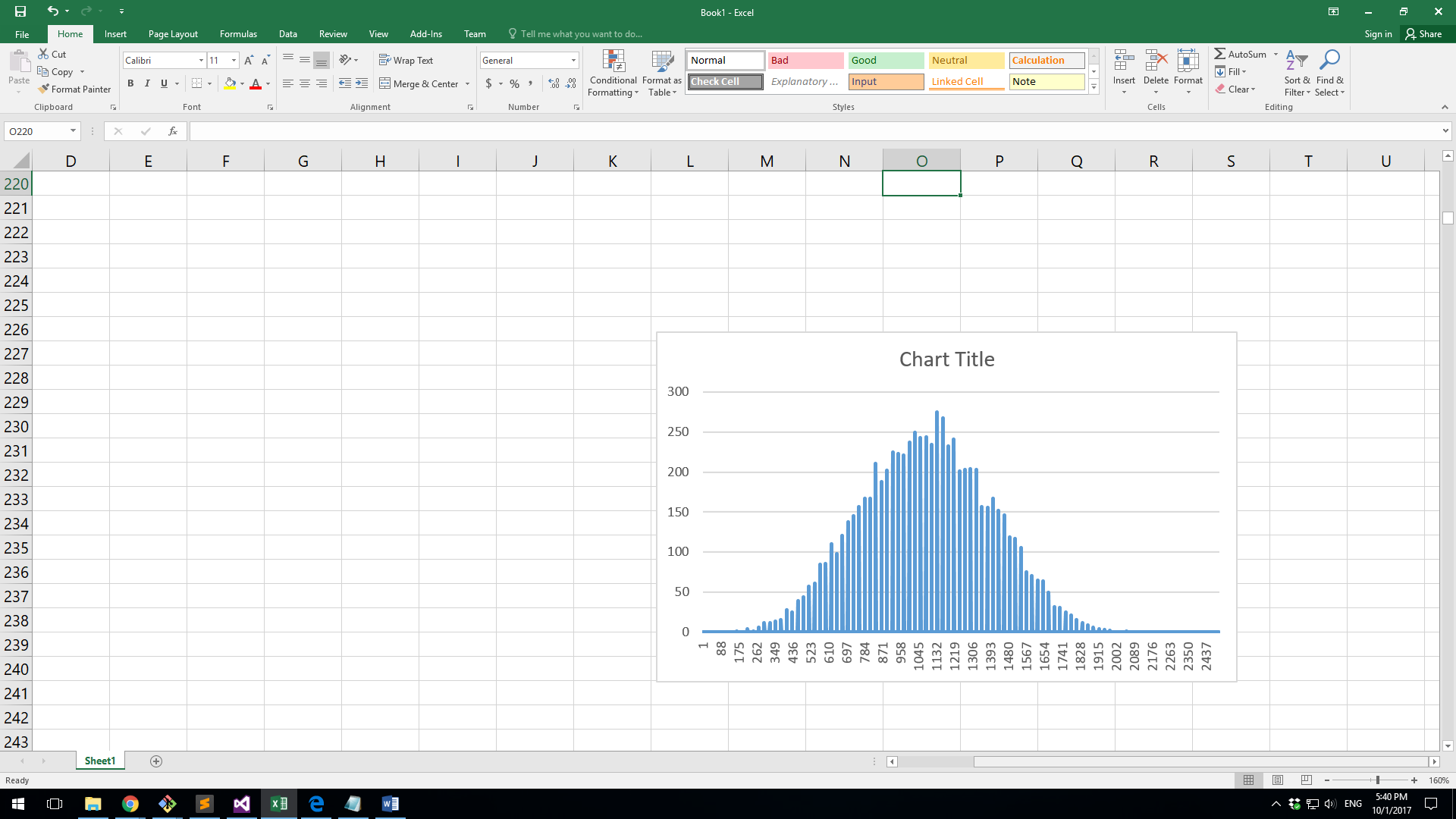
Для второй использовалась более хеш-функция, где просто сумма квадратов кодов символов.

**3. Результаты анализа хеш-функции**

В начале я использовал самую простую хеш-функцию

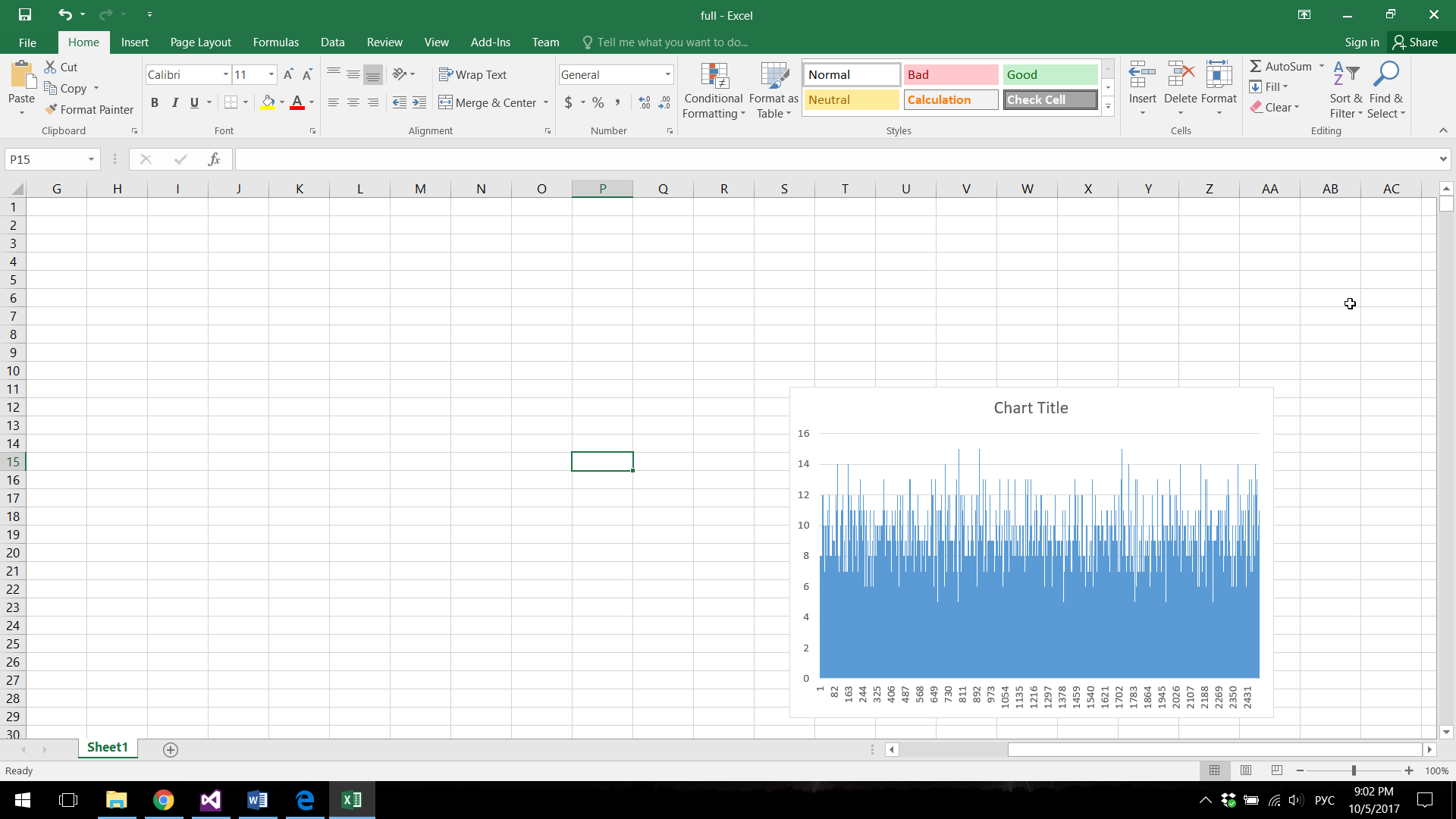
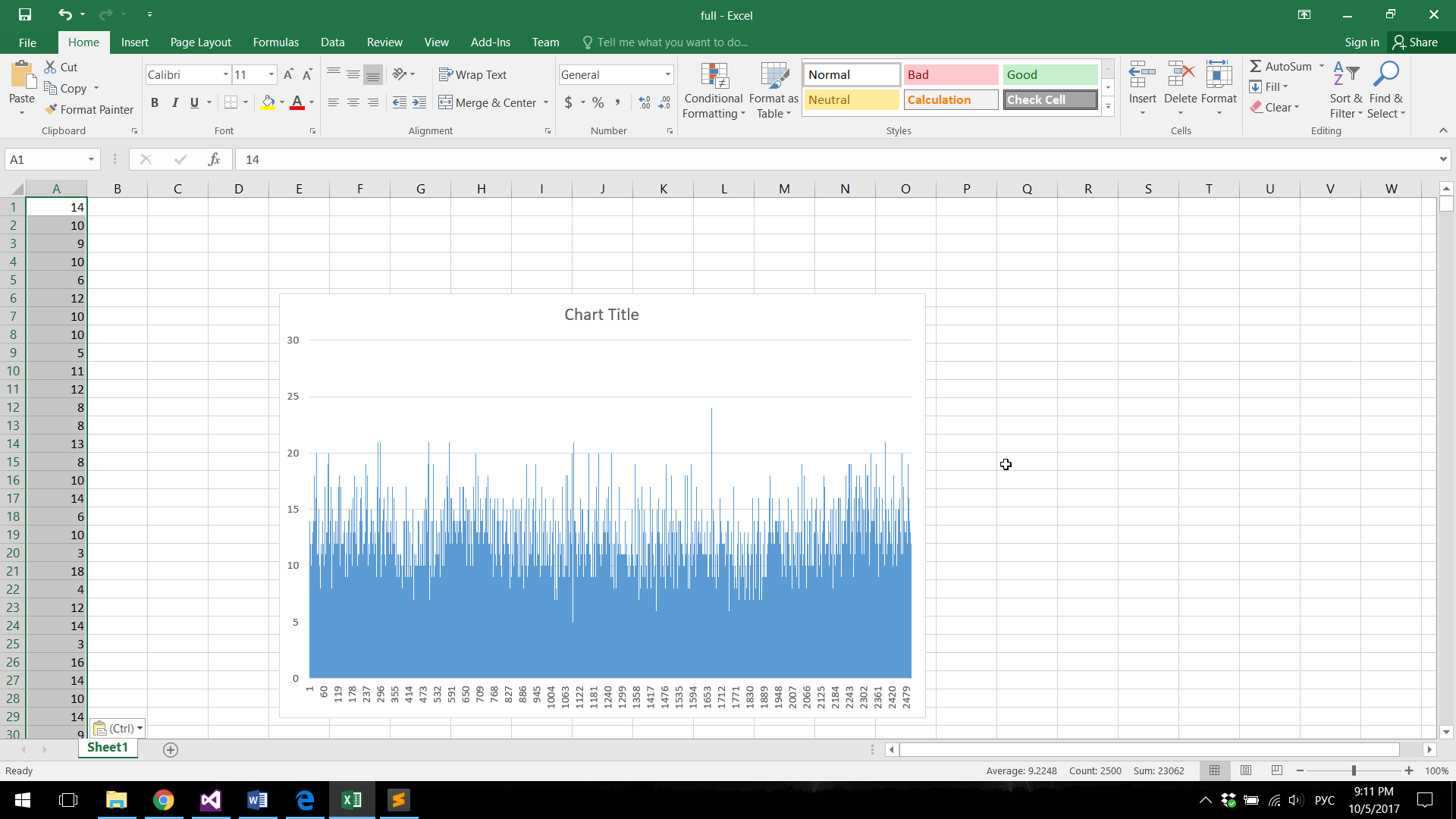
h(key) = (int)key[1] + (int)key[2] + (int)key[3] + (int)key[4] +(int)key[5] + (int)key[6]

Но график попадания ключей был не равномерен, для тестов я писал для 2500 элементов вычислял 7500 хешей и смотрел куда он укажет. Возведение в квадрат каждой *key[i]* решало эту проблему, но мне так же надо было создать и вторую хеш-функцию.

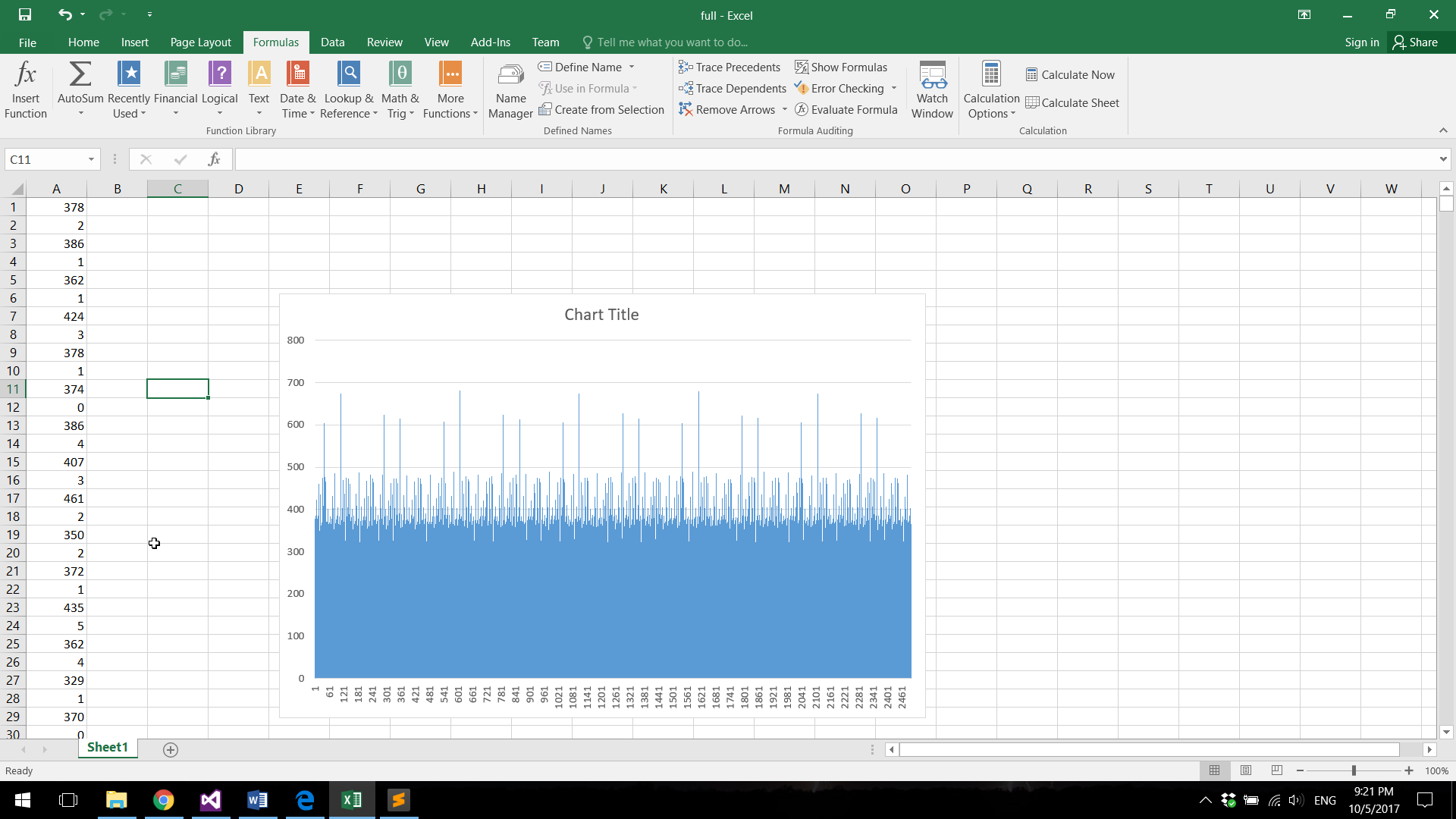


key[1] + key[2]… key[1]^2 + key[2] ^2…

Для первой хеш-функции я использовал полиномиальный хеш и если надо вычислять вторую, то используем key[1]^2 + key[2] ^2… для выбора простого числа. Простое число, чтобы при нахождении пустого элемента не попадать в одни и те же значения и максимальное кол-во итераций было известно заранее. Простые числа тоже вычисляются заранее до N и просто выбираются. N – число элементов в хеш-таблице. Итоговые графики кол-во обращений к элементу для заполнения всей таблицы:

Слева использование полиномиальонго хеша для, а справа использование для первой хеш-функции возведения в куб. Как видно, у первого случая в среднем пики меньше. В среднем у левого графика ~7.5, а у второго ~8.8. Так же если бы мы не использование простые числа, для h2, а просто возведение в квадрат или куб, то результат был бы таким:



**4. Листинг программы**

#include <iostream>

#include <string>

#include <time.h>

#include "HashMap.h"

using namespace std;

const int N = 2500;

const int K = 5;

void menu(HashMap<string> &HMap) {

cout << endl

<< "1 - Print\n"

<< "2 - Add element\n"

<< "3 - Delete elem\n"

<< "4 - Export\n"

<< "5 - Exit\n"

<< "Select the menu item: ";

int choice; cin >> choice;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

switch (choice) {

case 1: {

HMap.print();

menu(HMap);

break;

} case 2: {

string key;

string val;

cout << "Key: "; getline(cin, key);

if (key.length() == 6 && isupper(key[0]) && isupper(key[5]) &&

isdigit(key[1]) && isdigit(key[2]) && isdigit(key[3]) && isdigit(key[4])) {

cout << "Value: "; getline(cin, val);

HMap.put(key, val);

}

menu(HMap);

break;

} case 3: {

string key;

cout << "Key: "; getline(cin, key);

if (key.length() == 6 && isupper(key[0]) && isupper(key[5]) &&

isdigit(key[1]) && isdigit(key[2]) && isdigit(key[3]) && isdigit(key[4])) {

cout << "Elements with collisions: " << HMap.del(key).size() << endl;

}

menu(HMap);

break;

} case 4: {

HMap.excel(string("excel.txt"));

cout << "OK" << endl;

menu(HMap);

break;

} case 5: {

break;

} default:

cout << "Incorrect choice." << endl;

menu(HMap);

}

}

int main() {

srand(time(0));

HashMap<string> HMap(N);

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

string strKey(6, '0');

string strVal(10 + rand() % 70, '\0');

strKey[0] = 'A' + rand() % 26;

strKey[1] = '0' + rand() % 9;

strKey[2] = '0' + rand() % 9;

strKey[3] = '0' + rand() % 9;

strKey[4] = '0' + rand() % 9;

strKey[5] = 'A' + rand() % 26;

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

strVal[i] = (char) 'A' + rand() % 26;

HMap.put(strKey, strVal);

}

menu(HMap);

return 0;

}

#pragma once

#include <iostream>

#include <vector>

#include <fstream>

#include "HashNode.h"

using namespace std;

template <typename V>

class HashMap {

public:

HashMap(int N) {

MAP\_SIZE = N;

map = new HashNode<V> \*[N];

exprt = new int[N];

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

map[i] = nullptr;

exprt[i] = 0;

}

// Generate prime numbers for h2

// 2, 3, 5, 7, 11, 13, 17, 19...N

primes\_h2.push\_back(2);

for (int i = 3; i < N; i++) {

bool prime = true;

for (int j = 0; j < primes\_h2.size() && primes\_h2[j] \* primes\_h2[j] <= i; j++) {

if (i % primes\_h2[j] == 0) {

prime = false;

break;

}

}

if (prime)

primes\_h2.push\_back(i);

}

};

~HashMap() {

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

delete map[i];

delete map;

delete exprt;

};

unsigned int h1(string &key) {

unsigned int h1 = 0;

const int p = 37;

unsigned int p\_pow = 1;

// h(S) = S[0] + S[1] \* P + S[2] \* P ^ 2 + S[3] \* P ^ 3 + ... + S[N] \* P^N

for (size\_t i = 0; i < key.length(); i++) {

h1 += (key[i] - '0' + 1) \* p\_pow;

p\_pow \*= p;

}

return h1;

}

unsigned int h2(string &key) {

unsigned int h2 = 0;

for (size\_t i = 0; i < key.length(); i++)

h2 += key[i] \* key[i];

// Take a random item

return primes\_h2[h2 % primes\_h2.size()];

}

void put(string key, V value) {

unsigned int h1 = HashMap::h1(key);

unsigned int h2 = HashMap::h2(key);

for (size\_t i = 0; i < MAP\_SIZE; i++) {

exprt[h1 % N] += 1;

if (map[h1 % N] == nullptr) {

map[h1 % N] = new HashNode<V>(key, value);

return;

} else {

if (map[h1 % N]->getKey() == key)

map[h1 % N]->setValue(value);

h1 = (h1 + h2) % N;

}

}

cout << "OVERFLOW" << endl;

}

HashNode<V>\* seach(string &key) {

unsigned int h1 = HashMap::h1(key);

unsigned int h2 = HashMap::h2(key);

for (size\_t i = 0; i < MAP\_SIZE; i++) {

if (map[h1 % N] != nullptr && map[h1 % N]->getKey() == key)

return map[h1 % N];

else

h1 = (h1 + h2) % N;

}

return nullptr;

}

vector<HashNode<V>\*> del(string &key) {

unsigned int h1 = HashMap::h1(key);

unsigned int h2 = HashMap::h2(key);

vector<HashNode<V>\*> collision;

for (size\_t i = 0; i < MAP\_SIZE; i++) {

if (map[h1 % N]->getKey() == key) {

delete map[h1 % N];

map[h1 % N] = nullptr;

return collision;

} else {

collision.push\_back(map[h1 % N]);

h1 = (h1 + h2) % N;

}

}

}

void excel(string &name) {

ofstream fout(name);

for (size\_t i = 0; i < MAP\_SIZE; i++)

fout << exprt[i] << endl;

fout.close();

}

void print() {

cout << endl;

for (size\_t i = 0; i < MAP\_SIZE; i++)

if(map[i] != nullptr)

cout << i << ": "<< map[i]->getKey() << " - " << map[i]->getValue() << endl;

}

private:

HashNode<V> \*\*map;

int \*exprt;

int MAP\_SIZE;

vector<int> primes\_h2;

};

#pragma once

#include <string>

template <typename V>

class HashNode {

public:

HashNode(const std::string &key, const V &value):

key(key), value(value) {}

std::string getKey() const { return key; }

V getValue() const { return value; }

void setValue(V value) { HashNode::value = value; }

private:

std::string key;

V value;

};

**5. Вывод**

Во время работы выполнения лабораторной работы я познакомился с хеш-таблицами. Узнал много нового о них, о том лучше их делать и как делаю это такие компании как Goggle в google::dense\_hash\_map. Использовал vector для упрощения работы и так же ознакомился со string.