1. **Цель работы**

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

2. **Задание**

Согласно варианту №16:

16	ББццББ	2500	Двойное хеширование

3. **Описание хэш-функции**

   Хэш-функция представляет из себя сложение кодов символов.

$$h(k) = (int)key[0] + (int)key[1] + (int)key[2] + (int)key[3] + (int)key[4] + (int)key[5]$$

Чтобы хэш функция задевала все сегменты таблицы, необходимо вычесть 356(минимальный возможный хэш) и умножить на 21, таким образом хэш функция практически покрывает весь диапозон сегментов. ([0;2478])

4. **Анализ хэш-функции**

   После получения хэш-функции был проведён анализ коллизий и после первого тестирования была получена диаграмма на рисунке 1.

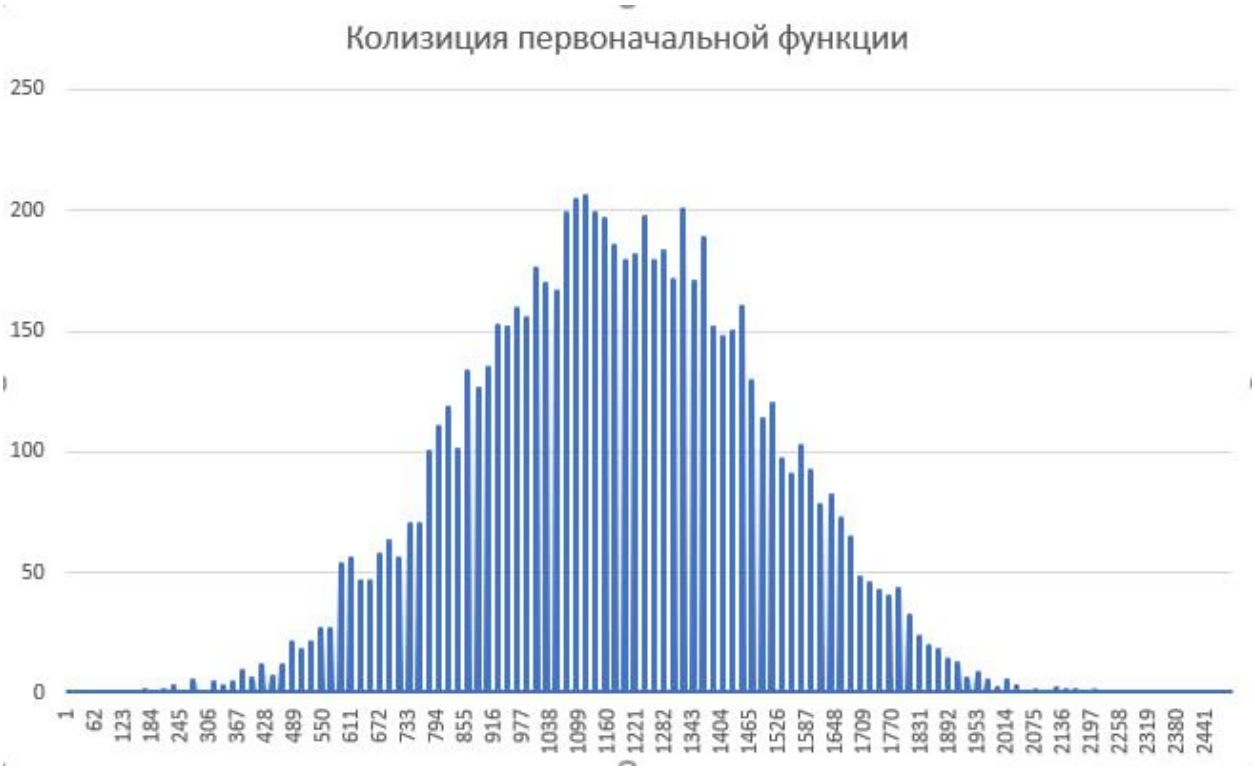

Рисунок 1 – коллизия первой функции

Было решено исправить хэш-функцию, чтобы исправить концетрацию в центре. Каждый код символа возводится в квадрат, а в конце берётся хэш по модулю всех сегментов. Таким образом была получена диаграмма на рисунке 2.

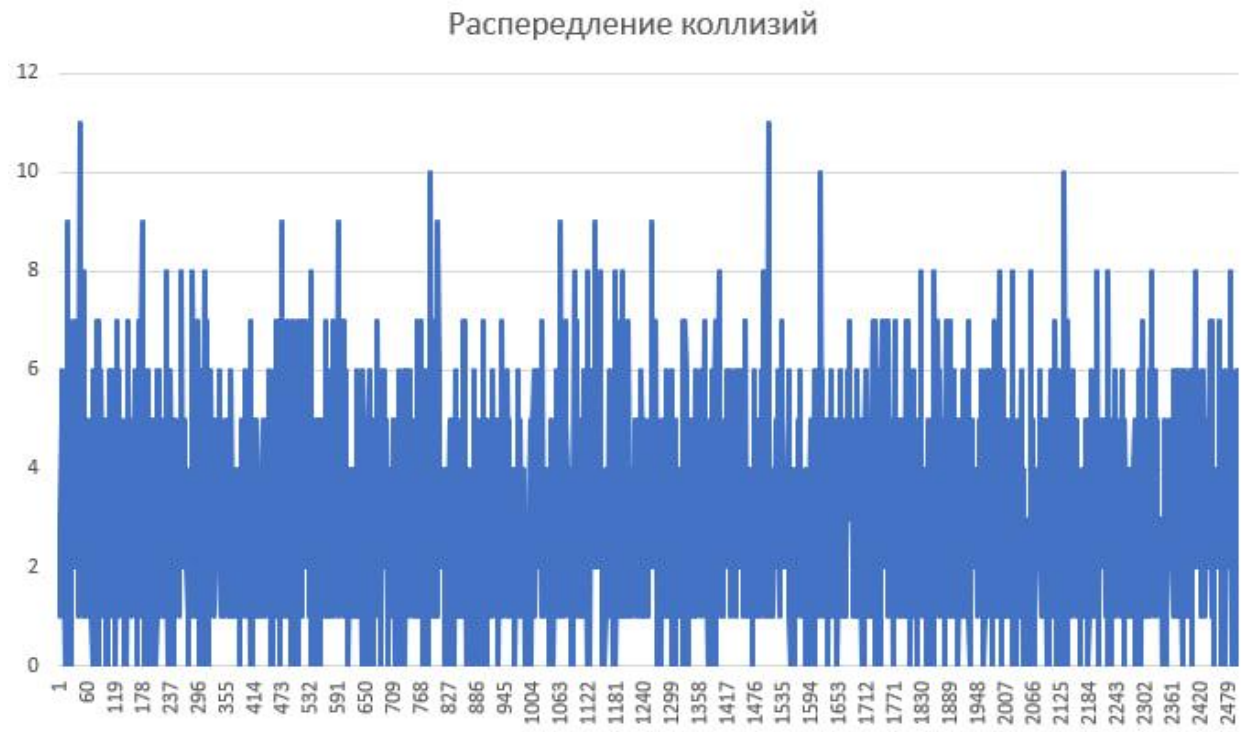

Рисунок 2 – Коллизия финальной функции

Как видно на диаграмме, концентрация распределена на весь промежуток, а значит хэш-функция построена верно.

5. Листинг программы

```
// Вариант 16
// Сегменты 2500
// Формат ББццББ

#include <iostream>
#include <string>
#include <fstream>

using namespace std;

int getHash(string key)
{
    int h = 0;
    for (int i = 0; i < key.length(); i++)
    {
        h += pow(key[i], 2);
    }
    h = (h - 356) * 21;
    h = h % 2500;
    return h;
}

void randomKey(char key[7], int m)
{
    for (int i = 0; i < m; i++) {
        if (i == 2 || i == 3)
            key[i] = '0' + rand() % (('9') - ('0'));
        else
            key[i] = 'A' + rand() % ('Z' - 'A');
    }
```

```
    key[6] = '\0';
}

void resultsToFile(int keys[2500], int seg)
{
    ofstream out("stats.txt");
    if (not(out.is_open()))
    {
        cout << "Не удаётся открыть файл" << endl;
        return;
    }
    for (int i = 0; i < seg; i++)
    {
        out << keys[i] << endl;
    }

    out.close();
}

void tabToFile(char keys[2500][7], int seg)
{
    ofstream out("hash.txt");
    if (not(out.is_open()))
    {
        cout << "Не удаётся открыть файл" << endl;
        return;
    }
    for (int i = 0; i < seg; i++)
    {
        out << i << ": " << keys[i] << endl;
    }

    out.close();
}

bool isFull(char tab[2500][7], int seg)
{
    for (int i = 0; i < seg; i++)
    {
        if (strcmp(tab[i], "\0") == 0)
        {
            return false;
        }

    }
    return true;
}


int findHashByKey(char tab[2500][7], char c[7], int seg)
{
    for (int i = 0; i < seg; i++)
    {
        if (strcmp(tab[i], c) == 0)
        {
            return i;
        }

    }
    return -1;
}

int findHash(char tab[2500][7], char c[7], int seg)
{
    int HASH = getHash(c);
```

```
    if (tab[HASH][0] != '\0')
    {
        return HASH;
    }
    return -1;
}

bool addHash(char tab[2500][7], char key[7], int seg)
{
    int HASH = getHash(key);
    if (isFull(tab, seg)) { return false; }
    if (findHashByKey(tab, key, seg) != -1) { return false; }
    if (findHash(tab, key, seg) == -1)
    {
        strcpy_s(tab[HASH], key);
        return true;
    }
    else
    {

        int HASH_2 = 1 + HASH % (seg - 1);
        long int k = 0;
        long int HASH_3 = (HASH + k * HASH_2) % seg;
        while (strcmp(tab[HASH_3], "\0") != 0)
        {
            k += 1;
            HASH_3 = (HASH + k * HASH_2) % seg;
            if (HASH_3 > seg || HASH_3 < 0)
            {
                return false;
                break;
            }
            if (k == seg)
            {
                return false;
                break;
            }
        }

        strcpy_s(tab[HASH_3], key);

        return true;
    }
}

bool delHash(char tab[2500][7], char c[7], int seg)
{
    for (int i = 0; i < seg; i++) {
        if (strcmp(tab[i], c) == 0)
        {
            strcpy_s(tab[i], "\0");
            return true;
        }

    }
    return false;
}

int main()
{
    setlocale(LC_ALL, "RUS");
    srand(time(NULL));
    const int m = 6; // Длина ключа
    const int seg = 2500;
    int keys[seg]; // Коллизии
```

```
char tab[seg][m + 1]; // Хэш таблица
char key[7]; // Ключ
int choicer;

cout << "Добро пожаловать в программу хэширования." << endl;

for (int i = 0; i < seg; i++)
{
    keys[i] = 0;
    tab[i][0] = '\0';
}

while (true)
{
    start:
    cout << "Выберите дальнейшее действие:" << endl;
    cout << "1. Заполнить таблицу" << endl;
    cout << "2. Добавить элемент" << endl;
    cout << "3. Удалить элемент" << endl;
    cout << "4. Найти элемент" << endl;
    cout << "5. Вывести таблицу" << endl;
    cout << "6. Сохранить таблицу в файл" << endl;
    cout << "7. Сохранить количество коллизий в файл" << endl;
    cout << "8. Выход" << endl;

    cin >> choicer;

    if (choicer == 1)
    {
        system("cls");

        for (int i = 0; i < seg; i++)
        {
            keys[i] = 0;
            tab[i][0] = '\0';
        }
        cout << "Заполнение..." << endl;
        for (int i = 0; i < seg*2; i++)
        {
            randomKey(key, m);
            int HASH = getHash(key);
            keys[HASH]++;
            addHash(tab, key, seg);
        }
        cout << "Таблица заполнена." << endl;
    }
    else if (choicer == 2)
    {
        system("cls");
        cout << "Для выхода введите 0" << endl;
        cout << "Введите ключ: ";

        cin >> key;
        if (strcmp(key, "0") == 0)
            goto start;

        bool right;
        for (int i = 0; i < m; i++) {
            if (i == 2 || i == 3)
                if (isdigit(key[i]))
                {
                    right = true;
                }
                else
                {
```

```
                right = false;
                break;
            }
        else if (i == 0 || i == 1 || i == 4 || i == 5)
            if (isalpha(key[i]))
            {
                right = true;
            }
            else
            {
                right = false;
                break;
            }
        else
        {
            right = false;
            break;
        }
    }

    if (right)
    {
        if (findHashByKey(tab, key, seg) == -1)
        {
            if (addHash(tab, key, seg))
            {
                cout << "Добавлено" << endl;
            }
            else
            {
                cout << "Не удалось добавить." << endl;
            }
        }
        else
        {
            cout << "Данный ключ уже существует." << endl;
        }

    }
    else
    {
        cout << "Ошибочный формат" << endl;
    }

}
else if (choicer == 3)
{
    system("cls");
    cout << "Для выхода введите 0" << endl;
    cout << "Введите ключ: ", cin >> key;
    if (strcmp(key, "0") == 0)
        goto start;
    if (delHash(tab, key, seg))
    {
        cout << "Удалено" << endl;
    }
    else
    {
        cout << "Ошибка" << endl;
    }
}
else if (choicer == 4)
{
    system("cls");
    cout << "Для выхода введите 0" << endl;
```

```
            cout << "Введите ключ: ", cin >> key;
            if (strcmp(key, "0") == 0)
                goto start;
            if (findHashByKey(tab, key, seg) != -1)
            {
                cout << "Позиция: " << findHashByKey(tab, key, seg) << endl;
            }
            else
            {
                cout << "Не найдено." << endl;
            }
        }
        else if (choicer == 5)
        {
            system("cls");
            cout << "Хэш-таблица:" << endl;
            for (int i = 0; i < seg; i++)
            {
                cout << i << " : " << tab[i] << endl;
            }
        }
        else if (choicer == 6)
        {
            system("cls");
            tabToFile(tab, seg);
            cout << "Таблица сохранена." << endl;
        }
        else if (choicer == 7)
        {
            system("cls");
            resultsToFile(keys, seg);
            cout << "Количество коллизий сохранено." << endl;
        }
        else if (choicer == 8)
        {
            cout << "Заверщение работы..." << endl;
            break;
        }
        else
        {
            system("cls");
        }
    }
}
```

6. Выводы

В процессе лабораторной работы были изучены методы хеширования и получены практические навыки реализации хэш-таблиц.