МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №3

по дисциплине
«Структуры и алгоритмы обработки данных»

Тема. Хеширование

Выполнил студент группы ИКБО-25-22 Ракитин В.А.

Принял доцент Бузыкова Ю.С.

Москва 2023

Цель: обобщение и систематизация знаний алгоритмов хеширования

Ход работы

1. Условие задания

- Соберите программу из фрагментов.
- Опишите назначение выбора 1 из меню программы.
- Реализуйте метод середины квадрата для хеш-функции. Допишите соответствующую опцию меню.
- Разработайте функцию пользователя сохранения хеш-таблицы в файл HashTable.txt

2. Тестовый пример

Копия содержания текстового файла представлена на рисунке 1.

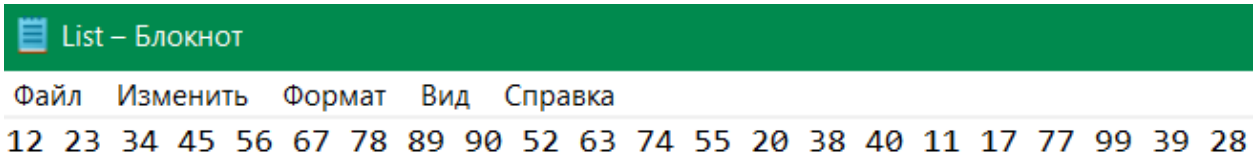

Рисунок 1 – Содержимое текстового файла

3. Код программы

Код программы представлен в листинге 1.

Листинг 1 – Код программы

```
#pragma warning(disable : 4996)
#include <stdio.h>
#include <conio.h>
#include <iostream>
#include <fstream>
#include <time.h>
using namespace std;


// Описание рекурсивной структуры данных - списка. В вершине хранится целое
число.
// Второй элемент структуры - указатель на список, который начнет
заполняться при
// разрешении коллизий
typedef struct Node_
{
    int data;
    struct Node_* next;
} Node;

// для удобства создаем определение типа int индекса в хеш-таблице
typedef int hashTableIndex;
```

```
int hashTableSize;

//Элементами хеш-таблицы по первому индексу являются указатели на
//вершину списка, соответствующего индекса. Двумерный массив создаем
//через типизированный указатель.
Node** hashTable1;

// хеш-функция метод деления элемента
// Key по модулю на размер таблицы HashTableSize
int Hash(int Key, int HashTableSize)
{
    return Key % HashTableSize;
}

// Другой вариант практически той же функции
// Хеш-функция размещения вершины для метода деления
hashTableIndex myhash1(int data)
{
    return (data % hashTableSize);
}

//Функция поиска местоположения и вставки вершины в таблицу для метода
цепочек
Node* insertNode_open(int data)
{
    Node* p, * p0;
    hashTableIndex bucket;

    //вставка вершины в начало списка
    bucket = myhash1(data);

    if ((p = new Node) == 0)
    {
        fprintf(stderr, "Нехватка памяти (insertNode)\n");
        exit(1);
    }

    p0 = hashTable1[bucket];
    hashTable1[bucket] = p;
    p->next = p0;
    p->data = data;

    return p;
}

//Функция удаления узла из таблицы для метода цепочек
void deleteNode_open(int data)
{
    Node* p0, * p;
    hashTableIndex bucket;
    p0 = 0;
    bucket = myhash1(data);
    p = hashTable1[bucket];

    while (p && !(p->data == data))
    {
        p0 = p;
        p = p->next;
    }

    if (!p)
        return;
```

```
    if (p0)
        p0->next = p->next;
    else
        hashTable1[bucket] = p->next;

    free(p);
}

//Функция поиска вершины со значением data для метода цепочек
Node* findNode_open(int data)
{
    Node* p;
    p = hashTable1[myhash1(data)];

    while (p && !(p->data == data))
        p = p->next;

    return p;
}

// Реализация хеш-функции методом середины квадрата
hashTableIndex myhash2(int data)
{
    int square = data * data;
    int middle = square / 100 % 10000;
    return middle % hashTableSize;
}

// Реализация функции вставки нового элемента в хеш-таблицу с коллизиями
void insertNodeSquare_open(int data)
{
    hashTableIndex index = myhash2(data);
        // Создаем новую вершину и заполняем ее данными
        Node* newNode = new Node;
        newNode->data = data;
        newNode->next = NULL;

    // Если в данном индексе еще нет вершин, добавляем новую
    if (hashTable1[index] == NULL)
    {
        hashTable1[index] = newNode;
    }
    // Если уже есть вершины, добавляем новую в начало списка
    else
    {
        Node* temp = hashTable1[index];
        hashTable1[index] = newNode;
        newNode->next = temp;
    }
}

// Реализация функции поиска вершины в хеш-таблице
void findNodeSquare_open(int data)
{
    hashTableIndex index = myhash2(data);
        // Если в данном индексе нет вершин, значит элемент не найден
        if (hashTable1[index] == NULL)
        {
            cout << "Element " << data << " not found" << endl;
        }
    // Иначе проходим по списку и ищем элемент
        else
        {
```

```
            Node* temp = hashTable1[index];
            while (temp)
            {
                if (temp->data == data)
                {
                    cout << "Element " << data << " found" << endl;
                    return;
                }
                temp = temp->next;
            }
            cout << "Element " << data << " not found" << endl;
        }
}

// Реализация функции удаления вершины из хеш-таблицы
void deleteNodeSquare_open(int data)
{
    hashTableIndex index = myhash2(data);
        // Если в данном индексе нет вершин, значит элемент не найден
        if (hashTable1[index] == NULL)
        {
            cout << "Element " << data << " not found" << endl;
        }
    // Если первая вершина в списке равна искомой, удаляем ее
        else if (hashTable1[index]->data == data)
        {
            Node* temp = hashTable1[index];
            hashTable1[index] = hashTable1[index]->next;
            delete temp;
            cout << "Element " << data << " deleted" << endl;
        }
    // Иначе проходим по списку и ищем элемент для удаления
        else
        {
            Node* prev = hashTable1[index];
            Node* curr = hashTable1[index]->next;
            while (curr)
            {
                if (curr->data == data)
                {
                    prev->next = curr->next;
                    delete curr;
                    cout << "Element " << data << " deleted" << endl;
                    return;
                }
                prev = curr;
                curr = curr->next;
            }
            cout << "Element " << data << " not found" << endl;
        }
}

//Главная функция программы
int main(int argc, char* argv[])
{

    srand(time(NULL));
    setlocale(LC_ALL, "Russian");
    FILE* fp;

    if ((fp = fopen("List.txt", "r")) == NULL)
    {
        perror("List.txt");
```

```
        return 1;
    }

    int vibor;
    printf("\nВыберите метод хеширования из списка:\n ");
    printf("\n 1. Метод деления по модулю");
    printf("\n 2. Метод середины квадрата");
    printf("\n\nМетоды обработки коллизий:\n ");
    printf("\n 3. Метод цепочек (открытое хеширование)");

    printf("\n\n Мой выбор: ");
    scanf_s("%d", &vibor);

    int i, * a, maxnum;

    switch (vibor)
    {
    case 1:
    {
        // Код для выбора первого метода (деление по модулю)
        int demo = 342;
        printf("\n\n Для key = %d функция возвращает (размер хеш-таблицы =
10): %d", demo, Hash(demo, 10));
        demo = 756;
        printf("\n Для key = %d функция возвращает (размер хеш-таблицы =
10): %d", demo, Hash(demo, 10));
        demo = 55556;
        printf("\n Для key=%d функция возвращает (размер хеш-таблицы = 10):
%d", demo, Hash(demo, 10));
        demo = 3412;
        printf("\n\n Для key=%d функция возвращает (размер хеш-таблицы =
100): %d", demo, Hash(demo, 100));
        demo = 7597;
        printf("\n Для key=%d функция возвращает (размер хеш-таблицы = 100):
%d", demo, Hash(demo, 100));
        demo = 55597;
        printf("\n Для key=%d функция возвращает (размер хеш-таблицы = 100):
%d", demo, Hash(demo, 100));
        break;
    }
    case 2:
    {
        printf("\n Введите количество элементов maxnum : ");
        scanf("%d", &maxnum);
        printf("\n Введите размер хеш-таблицы HashTableSize : ");
        scanf("%d", &hashTableSize);

        //Выделяем динамическую память для хеш-таблицы
        hashTable1 = new Node * [hashTableSize];

        //Задаем начальные значения элементам хеш-таблицы
        for (i = 0; i < hashTableSize; i++) {
            hashTable1[i] = NULL;
        }

        //Далее можно работать с файлом. Здесь массив используется
        //только для демонстрации работоспособности функций пользователя
        a = new int[maxnum];

        // Создаем переменные для работы с файлом
        int num;
        char c;
```

```
        // Считываем числа из файла и вставляем их в хеш-таблицу
        while (fscanf(fp, "%d%c", &num, &c) != EOF)
        {
            insertNodeSquare_open(num);
            if (c == '\n') break;
        }

        //Сохранение хеш-таблицы в файл HashTable.txt
        ofstream out("HashTable.txt");

        for (i = 0; i < hashTableSize; i++)
        {
            out << i << " : ";
            Node* Temp = hashTable1[i];

            while (Temp)
            {
                out << Temp->data << " -> ";
                Temp = Temp->next;
            }
            out << endl;
        }

        out.close();

        // Освобождаем память, удаляем хеш-таблицу
        for (int i = 0; i < hashTableSize; i++)
        {
            Node* temp = hashTable1[i];
            while (temp)
            {
                Node* prev = temp;
                temp = temp->next;
                delete prev;
            }
        }
        delete[] hashTable1;

        break;
    }
    case 3:
    {
        printf("\n Введите количество элементов maxnum : ");
        scanf("%d", &maxnum);
        printf("\n Введите размер хеш-таблицы HashTableSize : ");
        scanf("%d", &hashTableSize);

        //Выделяем динамическую память для хеш-таблицы
        hashTable1 = new Node * [hashTableSize];

        //Задаем начальные значения элементам хеш-таблицы
        for (i = 0; i < hashTableSize; i++)
            hashTable1[i] = NULL;

        //Далее можно работать с файлом. Здесь массив используется
        //только для демонстрации работоспособности функций пользователя
        a = new int[maxnum];

        //Считывание элементов массива из файла List.txt
        for (i = 0; i < maxnum; i++)
            fscanf(fp, "%d", &a[i]);

        fclose(fp);
```

```
        //Заполнение хеш-таблицы элементами массива
        for (i = 0; i < maxnum; i++)
        {
            insertNode_open(a[i]);
        }

        //Поиск элементов массива по хеш-таблице
        for (i = maxnum - 1; i >= 0; i--)
        {
            findNode_open(a[i]);
        }

        //Сохранение хеш-таблицы в файл HashTable.txt
        ofstream out("HashTable.txt");

        for (i = 0; i < hashTableSize; i++)
        {
            out << i << " : ";
            Node* Temp = hashTable1[i];

            while (Temp)
            {
                out << Temp->data << " -> ";
                Temp = Temp->next;
            }
            out << endl;
        }

        out.close();

        // Очистка хеш-таблицы
        for (i = maxnum - 1; i >= 0; i--)
        {
            deleteNode_open(a[i]);
        }

        break;
    }
    default:
    {
        printf("\n\n Неверный ввод!");
    }
    }

    printf("\n\n");
    system("pause");
    return 0;
}
```

4. Тестирование

Проверим работу метода деления по модулю. В результате выполнения работы метода, программа выведет ключи с их функциями. Результат работы метода представлен на рисунке 2.

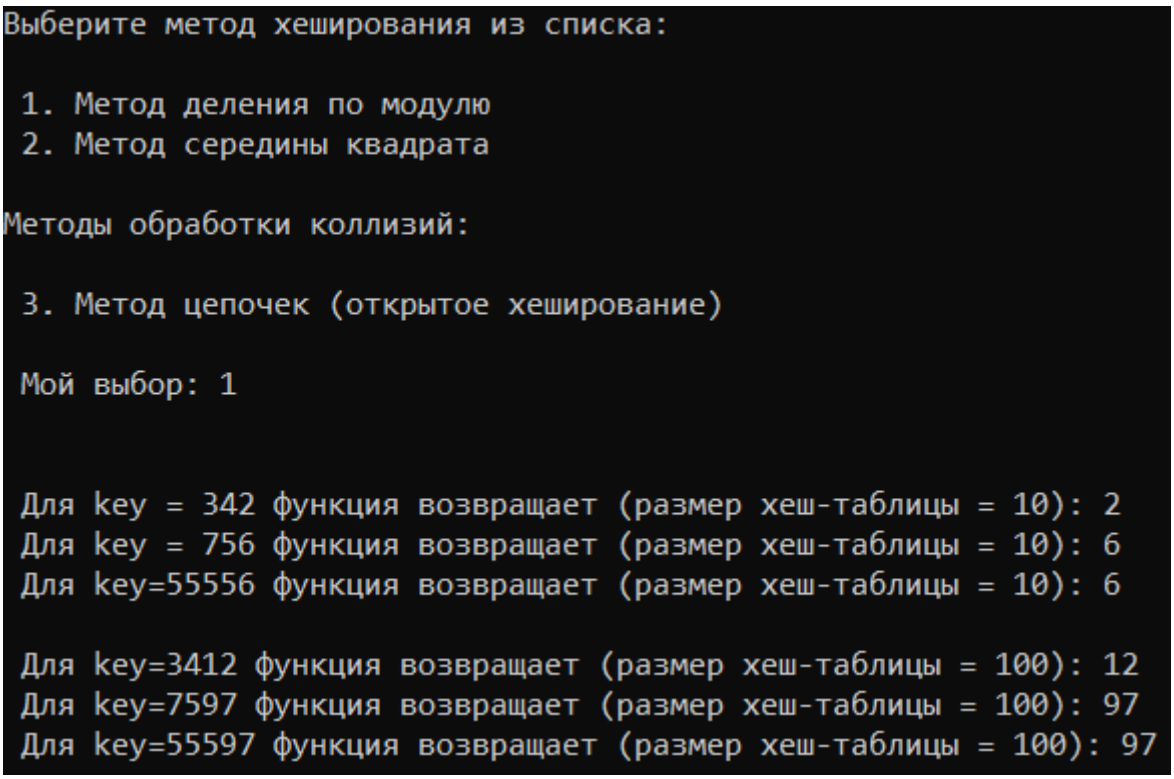

Рисунок 2 – Результат работы метода деления по модулю

Проверим работу метода середины квадрата. Создадим хеш-таблицу из 22 элементов и размером 6. Результаты работы метода и содержимое файла HashTable.txt представлены на рисунках 3 – 4.

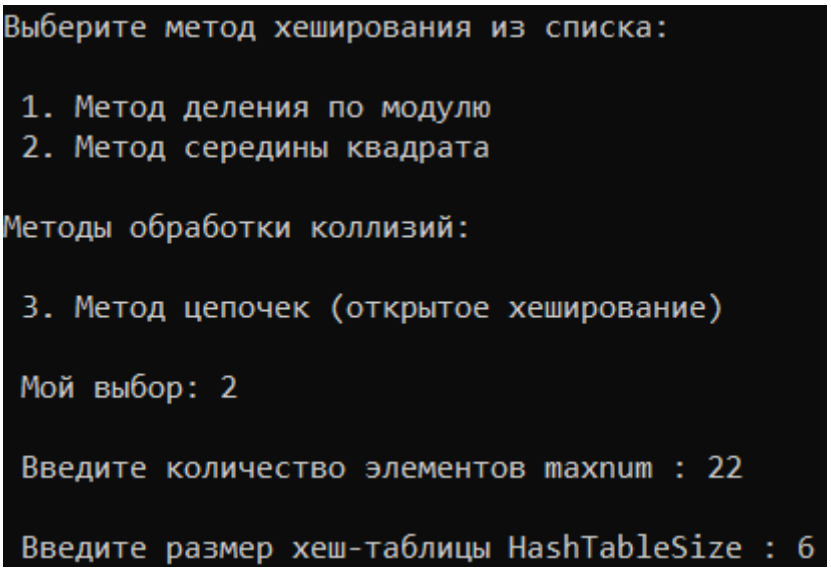

Рисунок 3 – Вызов метода середины квадрата

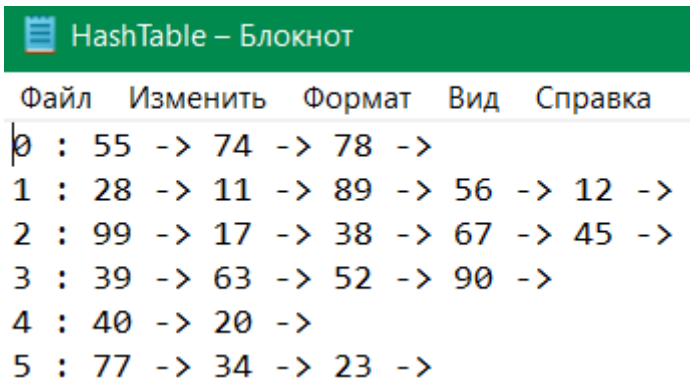

Рисунок 4 – Содержимое файла HashTable.txt после выполнения метода середины квадрата

Проверим работу метода цепочек (открытого хеширования). Создадим хеш-таблицу из 22 элементов и размером 6. Результаты работы метода и содержимое файла HashTable.txt представлены на рисунках 5 – 6.

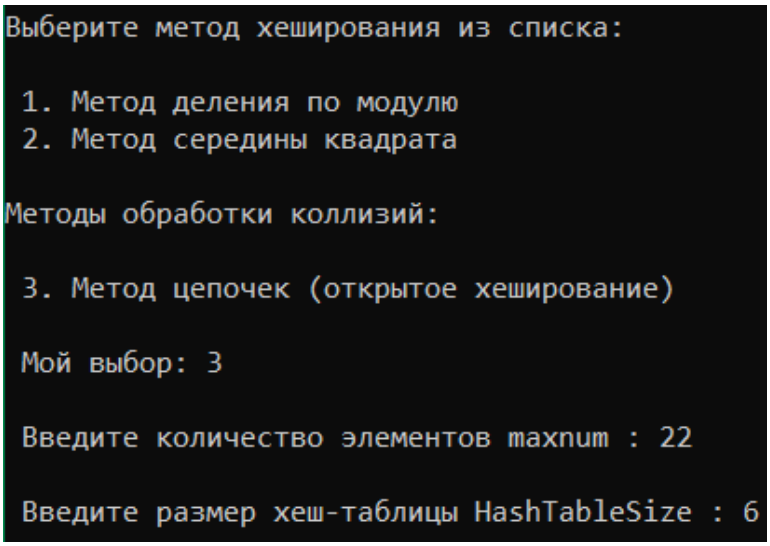

Рисунок 5 – Вызов метода цепочек

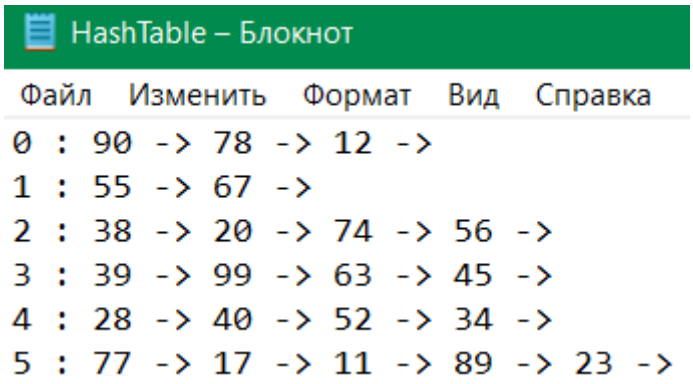

Рисунок 6 – Содержимое файла HashTable.txt после выполнения метода цепочек

Вывод: во время выполнения работы были получены навыки применения методов хеширования.