МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания № 6, часть 1

Тема:

«Быстрый доступ к данным с помощью хеш-таблиц»

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Лисовский И.В

Группа: ИКБО-21-23

Москва – 2024

СОДЕРЖАНИЕ

1 ЦЕЛЬ

Изучение и реализация механизма быстрого доступа к данным с использованием хеш-таблиц, позволяющего эффективно решать задачи поиска, вставки и удаления элементов с минимальной временной сложностью.

2 ЗАДАНИЕ

2.1 Формулировка задачи

Разработайте приложение, которое использует хеш-таблицу (пары «ключ - хеш») для организации прямого доступа к элементам динамического множества полезных данных. Множество реализуйте на массиве, структура элементов (перечень полей) которого приведена в индивидуальном варианте.

Приложение должно содержать класс с базовыми операциями: вставки, удаления, поиска по ключу, вывода. Включите в класс массив полезных данных и хеш-таблицу. Хеш-функцию подберите самостоятельно, используя правила выбора функции.

Реализуйте расширение размера таблицы и рехеширование, когда это требуется, в соответствии с типом разрешения коллизий.

Предусмотрите автоматическое заполнение таблицы 5-7 записями.

Реализуйте текстовый командный интерфейс пользователя для возможности вызова методов в любой произвольной последовательности, сопроводите вывод достаточными для понимания происходящего сторонним пользователем подсказками.

Проведите полное тестирование программы (все базовые операции, изменение размера и рехеширование), тест-примеры определите самостоятельно. Результаты тестирования включите

в отчет по выполненной работе.

Вариант №19:

19	Цепное хеширование	Книга: ISBN – 12-значное число, автор, название

2.2 Математическая модель решения

Хеш-таблица — это структура данных, которая используется для организации **быстрого доступа** к элементам на основе хеш-функции. Основной задачей в данном случае является эффективная вставка, удаление и поиск элементов множества данных. В данной математической модели будет

рассмотрен процесс организации множества данных с использованием хеш-таблицы и динамической переалокации при переполнении.:

Множество данных — это динамический массив элементов. Каждый элемент множества имеет следующие атрибуты: isbn - ключ типа long long int, который является уникальным идентификатором элемента; name - строка, представляющая имя элемента; author - строка, представляющая автора элемента.

Хеш-функция: Хеш-функция — это отображение ключа в индекс массива. Была использована формула h(k) = k mod n, где k – ключ элемента, n – длина хеш-таблицы.

Хеш-таблица состоит из 3-х полей: вместимости, количества элементов, и массива векторов для цепного хеширования.

```cpp
#include <iostream>
#include <vector>
#include <Windows.h>
using namespace std;

typedef long long int KEY;

struct item
{
    KEY isbn;
    string name;
    string author;
};

struct Hash_Table
{
    int length;
    int count;
    vector<item>* chains; // Массив векторов для цепочек
};

int hash_func(KEY key_, Hash_Table hash_table)
{
    return key_ % hash_table.length;
}
```

Листинг 1 — Структура записей, структура хеш-таблицы и хеш-функция

Цепное хеширование: В случае возникновения коллизий (одинаковый хеш для нескольких элементов), применяется **цепное хеширование**. Это означает, что в каждом индексе таблицы хранится вектор (динамический массив) элементов, которые имеют одинаковый хеш. Доступ к элементам в каждом векторе осуществляется через последовательный поиск.

Операция вставки в хеш-таблицу. При вставке элемента, вычисляется индекс – формируется с помощью хеш-функции. Далее, в массив элементов хеш-таблицы вставляется добавляемый элемент, при чем идет добавление в вектор с индексом рассчитанной хеш-функции.

```
void insertInHash(item example, Hash_Table& hash_table)
{
    int index = hash_func(example.isbn, hash_table);
    hash_table.chains[index].push_back(example);
    hash_table.count++;
    if (hash_table.count != 0)      //если таблица заполнена на 75% или больше, увеличиваем её
    {
        if ((float)(hash_table.count) / (float)(hash_table.length) >= 0.75)
        {
            hash_table = rehash_table(hash_table);
        }
    }

}
```

Листинг 2 —Код операции вставки

Операция рехеширования. Так как хеш-таблица будет пополнятся различными данными, то на нее будет большая нагрузка. В связи с этим, нужно помнить про условие, что если это коэффициент > 0.75, то следует увеличить вместимость хеш-таблицы в 2 раза.

```
Hash_Table rehash_table(Hash_Table& hash_table)
{
    Hash_Table new_table;       //новая хеш-таблица
    new_table.chains = new vector<item>[hash_table.length*2];   //выделяем место для цепей в два раза больше
    new_table.count = hash_table.count;         //переносим данные с прошлой таблицы
    new_table.length = hash_table.length * 2;
    for (int i = 0; i < hash_table.length; i++)
    {
        for (auto& it : hash_table.chains[i])
        {
            int index = hash_func(it.isbn, new_table);  //пересчитываем индекс каждого элемента относительно новой длины
            new_table.chains[index].push_back(it);
            new_table.count++;
        }
    }

    delete[] hash_table.chains;     //освобождение озу
    return new_table;
}
```

Листинг 3 —Код операции рехеширования

Операция поиска элемента. Для нахождения элемента, вычисляется хеш – номер цепочки, в которой лежит элемент. Внутри идет линейный поиск записи – сверяется исходный ключ с ключом записи.

```
void find_data(Hash_Table& hash_table, KEY to_find)
{
    int index = hash_func(to_find, hash_table);     //применяем хеш функцию для нахождения индекса
    for (auto& it : hash_table.chains[index])   //перебор по цепочке индекса index
    {
        if (it.isbn == to_find)     //если нашли элемент, то выводим его данные
        {
            cout << "isbn: " << it.isbn << "   author: " << it.author << "    name: " << it.name << "\n";
        }
    }
    cout << "not found\n";
}
```

Листинг 4 —Код операции поиска по хеш-таблице

Операция удаления элемента. Вычисляется хеш. Находится цепочка в которой лежит запись, по ней линейно ищется необходимая запись, которая в последствии удаляется из вектора.

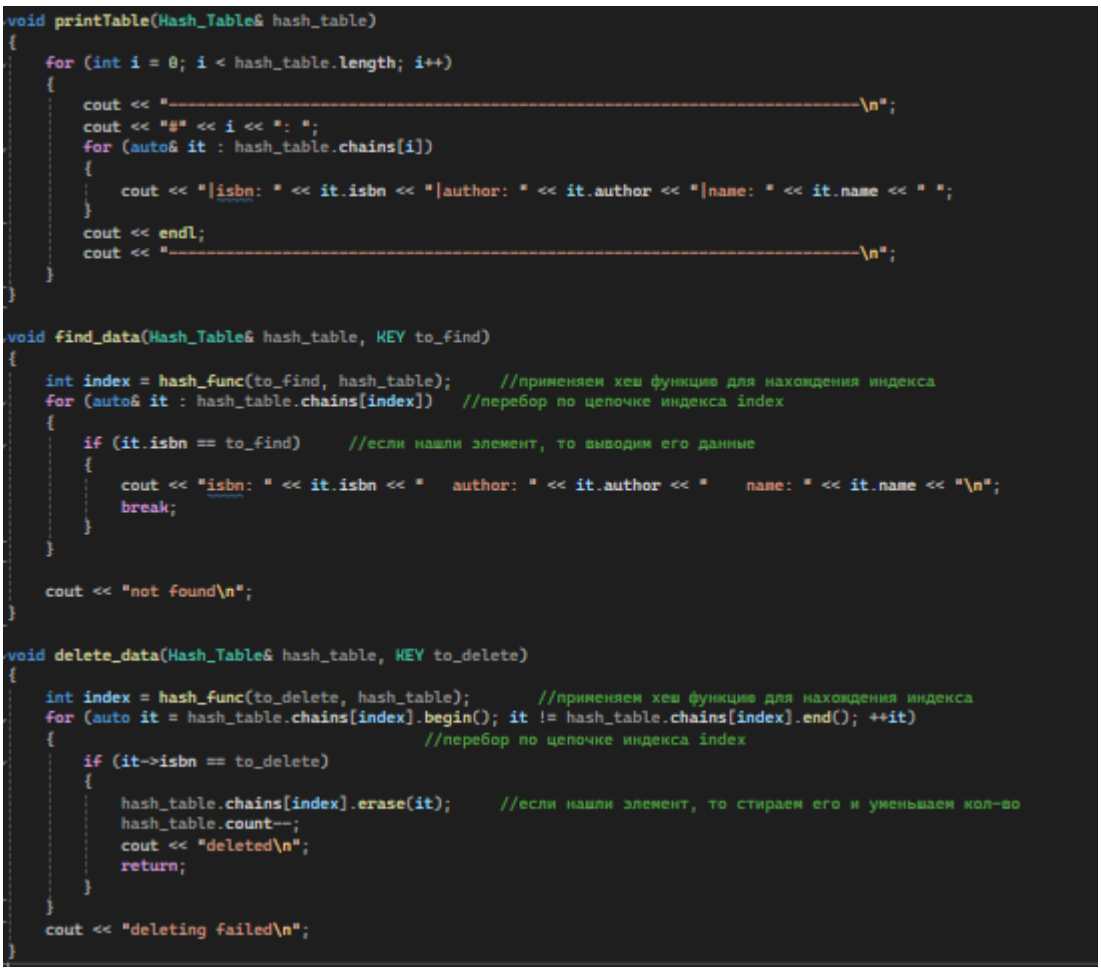
```
void delete_data(Hash_Table& hash_table, KEY to_delete)
{
    int index = hash_func(to_delete, hash_table);       //применяем хеш функцию для нахождения индекса
    for (auto it = hash_table.chains[index].begin(); it != hash_table.chains[index].end(); ++it)
    {                                       //перебор по цепочке индекса index
        if (it->isbn == to_delete)
        {
            hash_table.chains[index].erase(it);     //если нашли элемент, то стираем его и уменьшаем кол-во
            hash_table.count--;
            cout << "deleted\n";
            return;
        }
    }
    cout << "deleting failed\n";
}
```

Листинг 5 —Код операции удаления элемента

2.3 Код программы с комментариями

```cpp
#include <iostream>
#include <vector>
#include <Windows.h>
using namespace std;

typedef long long int KEY;

struct iten
{
    KEY isbn;
    string name;
    string author;
};

struct Hash_Table
{
    int length;
    int count;
    vector<item>* chains; // Массив векторов для цепочек
};

int hash_func(KEY key_, Hash_Table hash_table)
{
    return key_ % hash_table.length;
}

Hash_Table rehash_table(Hash_Table& hash_table)
{
    Hash_Table new_table;       //новая хеш-таблица
    new_table.chains = new vector<item>[hash_table.length*2];   //выделяем место для цепей в два раза больше
    new_table.count = hash_table.count;         //переносим данные с прошлой таблицы
    new_table.length = hash_table.length * 2;
    for (int i = 0; i < hash_table.length; i++)
    {
        for (auto& it : hash_table.chains[i])
        {
            int index = hash_func(it.isbn, new_table);  //пересчитываем индекс каждого элемента относительно новой длины
            new_table.chains[index].push_back(it);
        }
    }

    delete[] hash_table.chains;     //освобождение озу
    return new_table;
}

void insertInHash(item example, Hash_Table& hash_table)
{
    int index = hash_func(example.isbn, hash_table);
    hash_table.chains[index].push_back(example);
    hash_table.count++;
    if (hash_table.count != 0)      //если таблица заполнена на 75% или больше, увеличиваем её
    {
        if ((float)(hash_table.count) / (float)(hash_table.length) >= 0.75)
        {
            hash_table = rehash_table(hash_table);
        }
    }

}
```

Листинг 1 — Код программы



```
void printTable(Hash_Table& hash_table)
{
    for (int i = 0; i < hash_table.length; i++)
    {
        cout << "-------------------------------------------------------------------------\n";
        cout << "#" << i << ": ";
        for (auto& it : hash_table.chains[i])
        {
            cout << "|isbn: " << it.isbn << "|author: " << it.author << "|name: " << it.name << " ";
        }
        cout << endl;
        cout << "-------------------------------------------------------------------------\n";
    }
}

void find_data(Hash_Table& hash_table, KEY to_find)
{
    int index = hash_func(to_find, hash_table);     //применяем хеш функцию для нахождения индекса
    for (auto& it : hash_table.chains[index])   //перебор по цепочке индекса index
    {
        if (it.isbn == to_find)     //если нашли элемент, то выводим его данные
        {
            cout << "isbn: " << it.isbn << "   author: " << it.author << "    name: " << it.name << "\n";
            break;
        }
    }

    cout << "not found\n";
}

void delete_data(Hash_Table& hash_table, KEY to_delete)
{
    int index = hash_func(to_delete, hash_table);       //применяем хеш функцию для нахождения индекса
    for (auto it = hash_table.chains[index].begin(); it != hash_table.chains[index].end(); ++it)
    {                                   //перебор по цепочке индекса index
        if (it->isbn == to_delete)
        {
            hash_table.chains[index].erase(it);     //если нашли элемент, то стираем его и уменьшаем кол-во
            hash_table.count--;
            cout << "deleted\n";
            return;
        }
    }
    cout << "deleting failed\n";
}
```

Листинг 2 — Код программы

```
int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    Hash_Table table;
    item ex;    //инициализированная структура, нужна для пользовательской операции
    item ex1, ex2, ex3, ex4, ex5, ex6, ex7;
    ex1.author = "А.С.Пушкин";  ex1.isbn = 499558691032;    ex1.name = "Капитанская дочка";
    ex2.author = "М.Ю.Лермонтов"; ex2.isbn = 103583986392;  ex2.name = "Герой нашего времени";
    ex3.author = "Л.Н.Толстой"; ex3.isbn = 960303553069;    ex3.name = "Война и Мир";
    ex4.author = "С.Я.Маршак"; ex4.isbn = 869327484683;     ex4.name = "Кошкин дом";
    ex5.author = "А.А.Ахматова"; ex5.isbn = 683929395816;   ex5.name = "Реквием";          //готовые записи для операции auto
    ex6.author = "И.С.Тургенев"; ex6.isbn = 384885939945;   ex6.name = "Отцы и дети";
    table.length = 10;      //изначальные параметры для массива: вместимость 10, количество элементов 0
    table.count = 0;
    table.chains = new vector<item>[table.length]; // Выделяем память под массив векторов
    while (true)
    {
        cout << "0. auto\n1. add\n2. find\n3. print\n4. delete\n";
        int n;
        cin >> n;
        switch(n)
        {
        case 0:
            insertInHash(ex1, table);
            insertInHash(ex2, table);   //выполняется вставка в таблицу
            insertInHash(ex3, table);
            insertInHash(ex4, table);
            insertInHash(ex5, table);
            insertInHash(ex6, table);
            cout << "ready\n";
            break;
        case 1:
            cout << "isbn: ";
            cin >> ex.isbn;
            cout << "author: ";
            cin >> ex.author;
            cout << "name: ";
            cin >> ex.name;
            insertInHash(ex, table);
            break;
        case 2:
            KEY key_to_find;
            cout << "isbn: ";       //создание ключа, по которому будет идти поиск
            cin >> key_to_find;
            find_data(table, key_to_find);
            break;
        case 3:
            printTable(table);      //вывод таблицы
            break;
        case 4:
            KEY key_to_delete;      //срздание ключа, по которому будет идти удаление
            cout << "isbn: ";
            cin >> key_to_delete;
            delete_data(table, key_to_delete);
            break;
        }
    }

    delete[] table.chains; //очищаем память
    return 0;
}
```

Листинг 3 — Код программы

2.4 Тестирование программы

```
0. auto
1. add
2. find
3. print
4. delete
0
ready
0. auto
1. add
2. find
3. print
4. delete
3
------------------------------------------------------------------------
#0:
------------------------------------------------------------------------
------------------------------------------------------------------------
#1:
------------------------------------------------------------------------
------------------------------------------------------------------------
#2: |isbn: 499558691032|author: А.С.Пушкин|name: Капитанская дочка |isbn: 103583986392|author: М.Ю.Лермонтов|name: Герой нашего времени
------------------------------------------------------------------------
------------------------------------------------------------------------
#3: |isbn: 869327484683|author: С.Я.Маршак|name: Кошкин дом
------------------------------------------------------------------------
------------------------------------------------------------------------
#4:
------------------------------------------------------------------------
------------------------------------------------------------------------
#5: |isbn: 384885939945|author: И.С.Тургенев|name: Отцы и дети
------------------------------------------------------------------------
------------------------------------------------------------------------
#6: |isbn: 683929395816|author: А.А.Ахматова|name: Реквием
------------------------------------------------------------------------
------------------------------------------------------------------------
#7:
------------------------------------------------------------------------
------------------------------------------------------------------------
#8:
------------------------------------------------------------------------
------------------------------------------------------------------------
#9: |isbn: 960303553069|author: Л.Н.Толстой|name: Война и Мир
------------------------------------------------------------------------
0. auto
1. add
2. find
3. print
4. delete
```

Рисунок 1 — Тестирование кода – автоматический ввод данных и вывод

Рисунок 2 — Тестирование кода – добавление элемента, создание коллизии

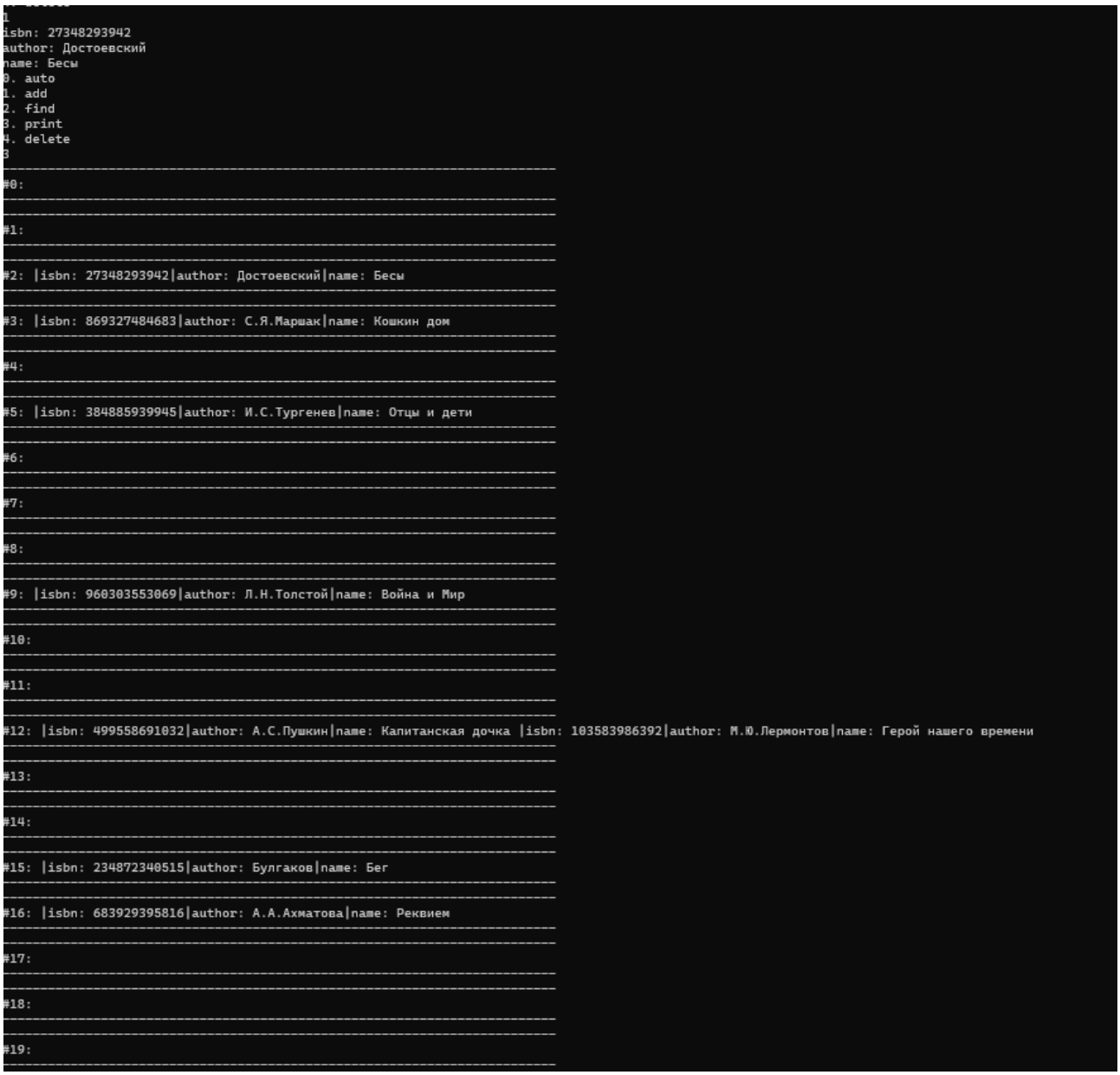


Рисунок 3 — Тестирование кода – создание нагрузки, в последствии - рехеширование

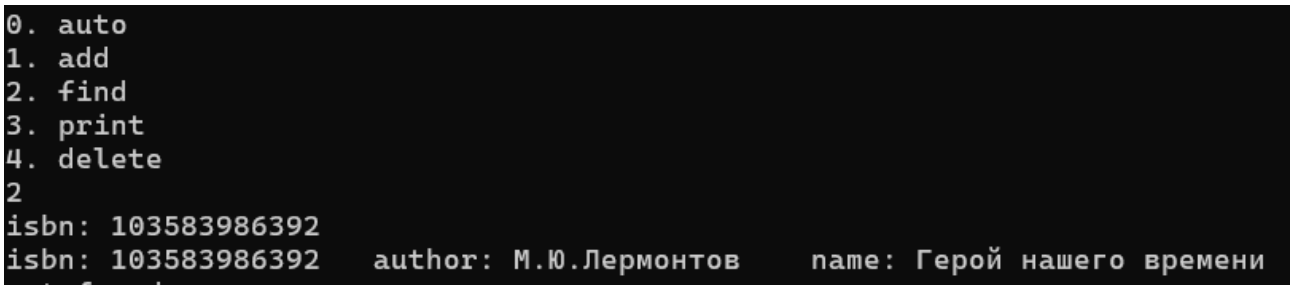


Рисунок 4 — Тестирование кода функции поиска по ключу

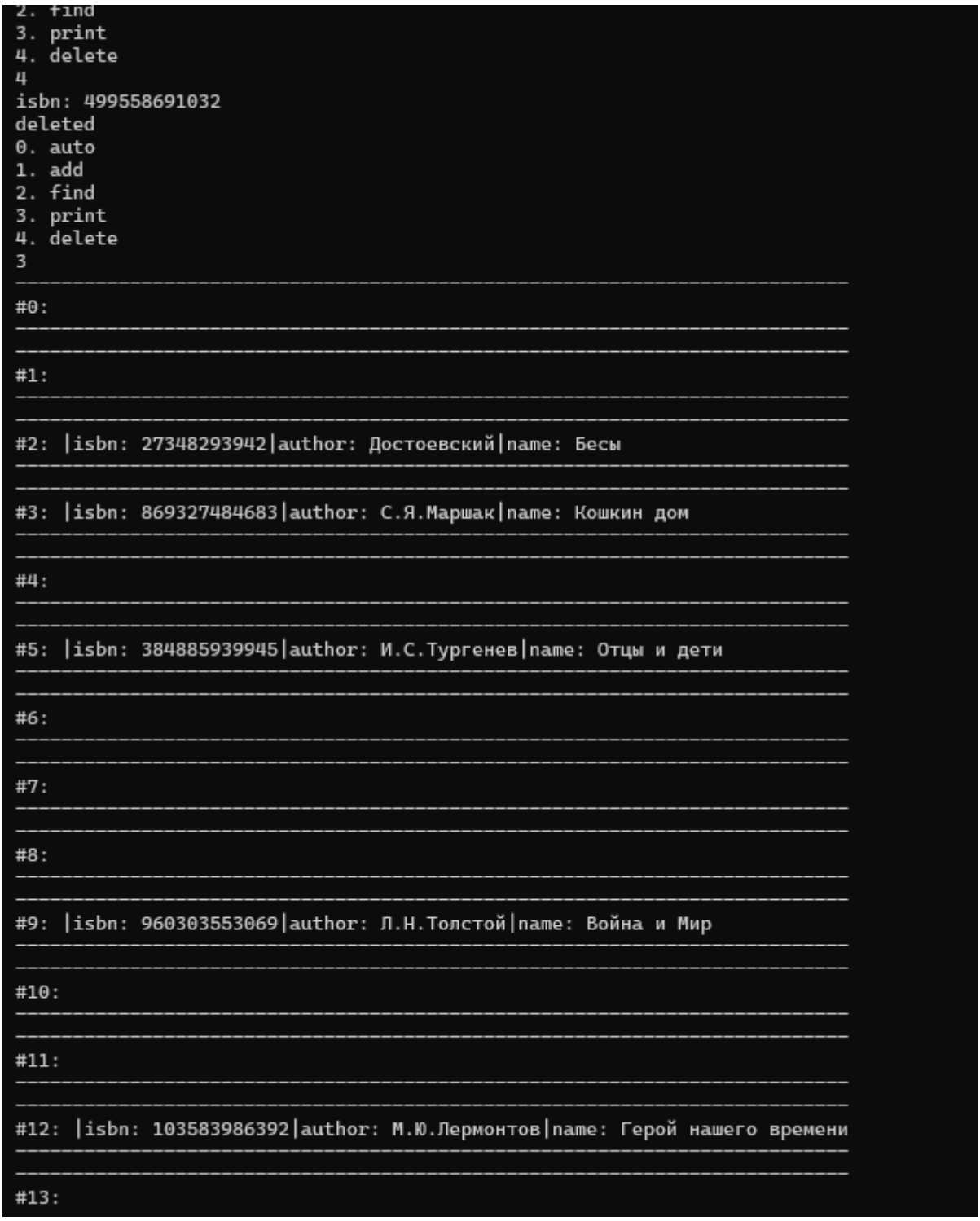


Рисунок 5 — Тестирование кода функции удаления по ключу

\

6 ВЫВОД

В ходе работы была реализована хеш-таблица с использованием метода цепного хеширования на основе векторов для организации быстрого доступа к данным. В процессе разработки были выполнены следующие задачи:

1. Разработана и реализована эффективная хеш-функция, которая использует остаток от деления ключа на размер таблицы, что позволяет равномерно распределять данные по хеш-таблице.
2. Введена структура данных, обеспечивающая хранение ключей и связанных с ними элементов, а также базовые операции для работы с хеш-таблицей: вставка, поиск, удаление и вывод данных.
3. Для обработки коллизий использовано цепное хеширование, где каждая ячейка хеш-таблицы хранит вектор элементов, попавших в одну и ту же корзину. Это позволяет эффективно решать проблему коллизий и поддерживать высокую производительность операций.
4. Реализован механизм перехеширования, который автоматически увеличивает размер хеш-таблицы при достижении порога заполненности (75%). Это обеспечивает баланс между временем вставки элементов и использованием памяти, предотвращая ухудшение производительности при увеличении количества элементов.
5. Операции вставки, поиска и удаления данных имеют амортизированную временную сложность O(1), что делает хеш-таблицы высокоэффективным инструментом для организации быстрого доступа к данным в больших наборах.

Реализованная программа демонстрирует высокую производительность при работе с большими объемами данных и решает задачу организации быстрого доступа к элементам множества. Хеш-таблица является отличным выбором для систем, требующих постоянного доступа к данным за минимальное время.

7 ЛИТЕРАТУРА

1. Вирт Н. Алгоритмы и структуры данных. Новая версия для Оберона, 2010.
2. Кнут Д. Искусство программирования. Тома 1-4, 1976-2013.
3. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих, 2017.
4. Кормен Т.Х. и др. Алгоритмы. Построение и анализ, 2013.
5. Лафоре Р. Структуры данных и алгоритмы в Java. 2-е изд., 2013.
6. Макконнелл Дж. Основы современных алгоритмов. Активный обучающий метод. 3-е доп. изд., 2018.
7. Скиена С. Алгоритмы. Руководство по разработке, 2011.
8. Хайнеман Д. и др. Алгоритмы. Справочник с примерами на C, C++, Java и Python, 2017.
9. Гасфилд Д. Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология, 2003.

По языку C++:

10. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.
11. Павловская Т.А. C/C++. Программирование на языке высокого уровня, 2003.
12. Прата С. Язык программирования C++. Лекции и упражнения. - 6-е изд., 2012.
13. Седжвик Р. Фундаментальные алгоритмы на C++, 2001-2002
14. Хортон А. Visual C++ 2010. Полный курс, 2011.
15. Шилдт Г. Полный справочник по C++. 4-е изд., 2006.