МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий
Кафедра математического обеспечения и стандартизации ИТ

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №2.2: Хеширование: прямой доступ к данным.
ПО ДИСЦИПЛИНЕ
« СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ»

Выполнил студент группы ИКБО-01-21 Маров Г.А.

Принял старший преподаватель Туманова М.Б.

Практическая работа выполнена «10» сентября 2022 г. _____ *(подпись студента)*

«Зачтено» « » сентября 2022 г. _____ *(подпись руководителя)*

Москва 2022

Цель: Освоить приёмы хеширования и эффективного поиска элементов множества.

Ход работы

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 (п.3). 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.

Индивидуальный вариант (24): Цепное хеширование. Товар: код – шестиразрядное число, название, цена.

Математическая модель решения: поскольку в качестве ключа выступает код товара (шестиразрядное число), и, согласно индивидуальному варианту, в качестве механизма разрешения коллизий необходимо использовать цепное хеширование, были выбраны хэш-функции, основанные на методе деления. Таким образом, в процессе выполнения работы была использована следующая хэш-функция:

$f(k) = k \bmod N,$

где k – ключ, а N – текущее максимальное кол-во записей массива.

Добавление записей выполняется в соответствии со стандартным алгоритмом цепного хэширования: хэш-функция на основе ключа вычисляет индекс ячейки хеш-таблицы, в которую попадает элемент и его ключ. В случае коллизии элемент, вызвавший ее, помещается следующим в связанном

списке по индексу ячейки, а предыдущий получает его адрес на хранение. Если массив и списки становятся не однородными, происходит рехеширование с увеличением размера таблицы в два раза.

Код программы с комментариями:

Header-файл:

```
#ifndef __HASH_H__
#define __HASH_H__
#include <iostream>
#include <vector>
#include <string>
#include <map>
using namespace std;

// структура для хранения информации о товарах
struct goods
{
	int code;
	string title;
	int cost;
	goods(int _code = NULL, string _title = "no such a product", int _cost = NULL) :
code(_code), title(_title), cost(_cost) {};
};

// структура узла
struct Node
{
	goods product;
	Node* next;
	Node(goods _product) : product(_product), next(nullptr) {};
};

// структура односвязного списка
struct list
{
    Node* first;
    Node* last;
    list() : first(nullptr), last(nullptr) {};
    bool is_empty();
       void push_back(goods product);
       void print();
       Node* find(int _code);
       Node* operator[](const int index);
       void delete_node(int code);
};

class Hash
{
	map<int, list> hash_table; // хеш таблица
	vector<goods> data; // important data storage
	vector<list> table; // buckets
public:
	int hash_func(int key); // хэш функция
	void rehash(int size); // функция рехеширования
	void update_data(goods product); // функция для добавления в базу данных
	void insert(goods product); // вставка в хэш таблицу
	void erase(int key); // удаление из хеш таблицы
	goods value(int key); // поиск по коду в хеш таблице
	void output(); // вывод хеш таблицы
```

```
};
#endif
```

Source-файл:

```
#include "Hash.h"

bool list::is_empty()
{
    return first == nullptr;
}
void list::push_back(goods product)
{
    Node* p = new Node(product);
    if (is_empty()) {
        first = p;
        last = p;
        return;
    }
    last->next = p;
    last = p;
}

Node* list::find(int _code)
{
    Node* p = first;
    while (p && p->product.code != _code) p = p->next; // пока не дойдем до нужного кода
следуем дальше
    return (p && p->product.code == _code) ? p : nullptr; // возвращаем указатель на
нужный товар
}

Node* list::operator[] (const int index)
{
    if (this->is_empty()) return nullptr; // если пустой лист то возвращаем пустой
указатель
    Node* p = first;
    for (int i = 0; i < index; i++) {
        p = p->next; // ищем товар с заданным индексом последовательно
        if (!p) return nullptr; // если не нашли то пустой указатель
    }
    return p; // возвращаем указатель на товар
}

void list::delete_node(int _code)
{
    int i = 0;
    int target = 0;
    Node* p = first;
    while (p && p->product.code != _code) p = p->next; // следуем по списку до нужного
кода
    (*this)[i - 1]->next = (*this)[i]->next; // меняем значения next предыдущего узла

    // удаляем выбранный товар
    p = nullptr;
    delete p;
}

int Hash::hash_func(int key)
{
    return key % table.size(); // хеш функция на делении
}

void Hash::rehash(int size)
{
```

```
    cout << "logging: table rehashed with size of " << size << endl;
    table.assign(size, list()); // делаем пустой массив заданного размера
    for (int i = 0; i < data.size(); i++) {
        table[hash_func(data[i].code)].push_back(data[i]); // добавляем в зависимости от
хеша в нужную ячейку списки
        hash_table.insert(make_pair(data[i].code, table[hash_func(data[i].code)])); //
добавляем в хеш таблицу хеш
    }
    if ((double)(data.size() / table.size()) > 0.75) // если не однородная хеш таблица то
увеличиваем размер в 2 раза и заново
        rehash(table.size() * 2);
}

void Hash::update_data(goods product)
{
    data.push_back(product); // добавляем в базу данных новый товар
}

void Hash::insert(goods product)
{
    table[hash_func(product.code)].push_back(product); // добавляем новый товар в массив
хешей
    this->update_data(product); // добавляем в базу данных
    hash_table.insert(make_pair(product.code, table[hash_func(product.code)])); //
добавляем в таблицу хеш
    if ((double)(data.size() / table.size()) > 0.75)
        rehash(table.size() * 2); // если не однородная хеш таблица то увеличиваем размер
в 2 раза и рехешируем
}

void Hash::erase(int key)
{
    table[hash_func(key)].delete_node(key); // удаляем узел с товаром
    hash_table.erase(key); // удаляем ключ
    for (int i = 0; i < data.size(); i++)
    {
        if (data[i].code == key)
            data.erase(data.begin() + i); // удаляем из базы данных
    }
}

goods Hash::value(int key)
{
    return hash_table[key].find(key)->product; // находим нужный товар в списке
}

void Hash::output()
{
    cout << "---- hash table ----" << endl;
    for (int i = 0; i < data.size(); i++)  // выводим все товары из базы данных
        if (data[i].code != NULL) {
            goods product = value(data[i].code);
            cout << "---- " << i + 1 << " product ----" << endl;
            cout << "code: " << product.code << endl;
            cout << "title: " << product.title << " | cost: " << product.cost << endl;
        }
    cout << "--------------------" << endl;
}
```

Main-функция:

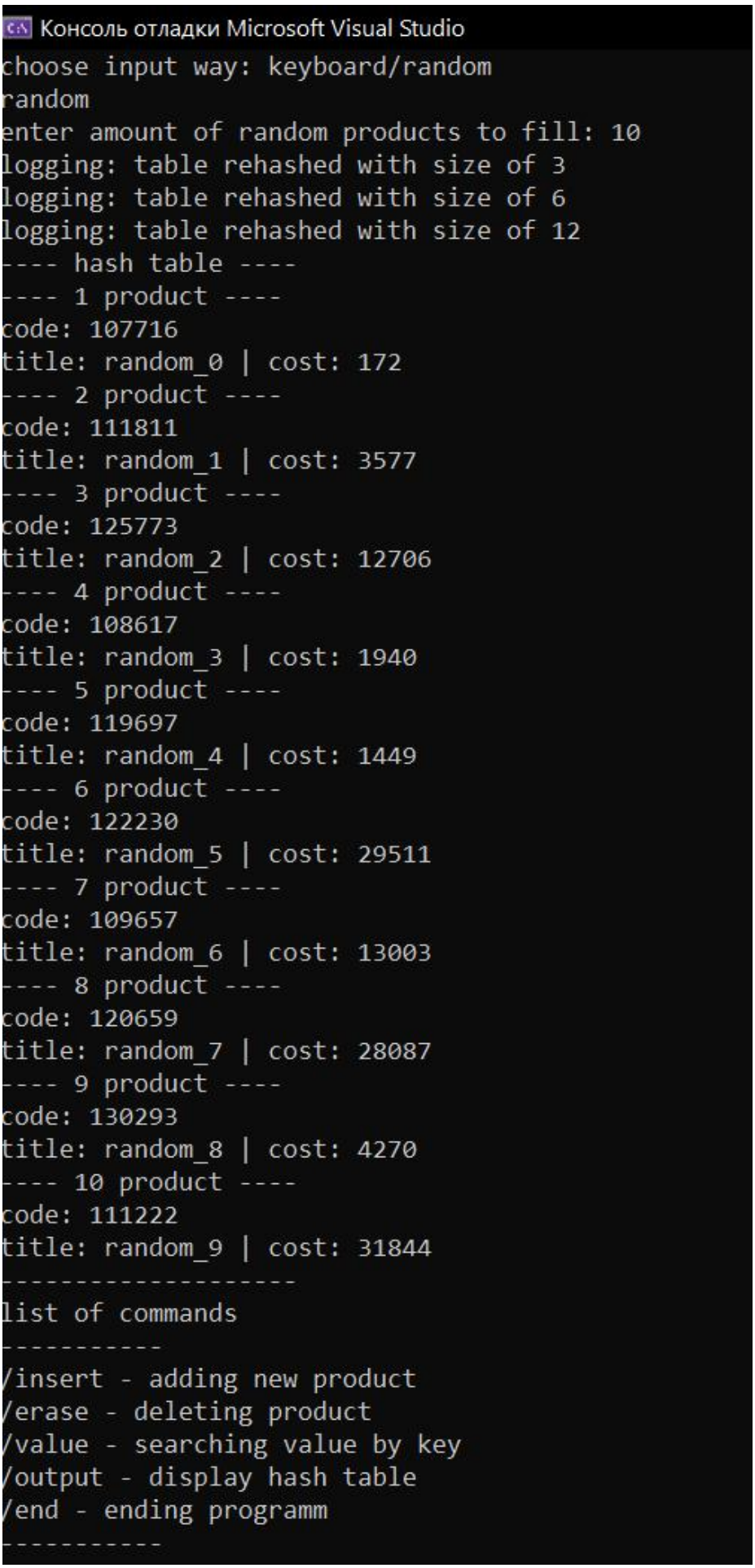
```
#include "Hash.h"
#include <iostream>
using namespace std;
```

```
int GetRandomNumber(int min, int max) // генератор случайного числа в заданном диапазоне
{
    unsigned rand_gen = rand();
    srand(rand_gen);
    return min + rand() % (max - min + 1);
}

int main()
{
    Hash ans;
    string command;
    cout << "choose input way: keyboard/random" << endl;

    // заполнение таблицы стартовыми значениями в зависимости от выбранного способа
    while(getline(cin, command)) {
        if (command == "random") {
            int amount;
            cout << "enter amount of random products to fill: ";
            cin >> amount;
            for (int i = 0; i < amount; i++) { // добавление в данные случайных товаров
                ans.update_data(goods(GetRandomNumber(100000, 999999), "random_" +
to_string(i), GetRandomNumber(0, 100000)));
            }
            break;
        }
        else if (command == "keyboard") {
            cout << "enter amount of goods: ";
            int n;
            cin >> n;
            for (int i = 0; i < n; i++)
            {
                int code;
                cout << "enter " << i + 1 << " product code: ";
                cin >> code;
                string title;
                cout << "enter " << i + 1 << " product title: ";
                cin >> title;
                int cost;
                cout << "enter " << i + 1 << " product cost: ";
                cin >> cost;
                ans.update_data(goods(code, title, cost)); // добавление в базу данных
введенного значения
            }
            break;
        }
        else
            cout << "wrong input tipe. try again" << endl;
    }
    ans.rehash(3); // стартовый массив хеш индексов
    ans.output();
    cout << "list of commands" << endl;
    cout << "-----------" << endl;
    cout << "/insert - adding new product" << endl;
    cout << "/erase - deleting product" << endl;
    cout << "/value - searching value by key" << endl;
    cout << "/output - display hash table" << endl;
    cout << "/end - ending programm" << endl;
    cout << "-----------" << endl;
    getline(cin, command);
    cout << "-> enter next command: ";

    // работа с командами
    while (getline(cin, command)) {
```

```
        if (command == "/insert") {
            cout << "-----------" << endl;
            int code;
            cout << "enter product code: ";
            cin >> code;
            string title;
            cout << "enter product title: ";
            cin >> title;
            int cost;
            cout << "enter product cost: ";
            cin >> cost;
            cout << "-----------" << endl;
            ans.insert(goods(code, title, cost)); // добавление введенного товара в хеш
таблицу
            getline(cin, command);
            cout << "-> enter next command: ";
        }
        else if (command == "/erase") {
            int code;
            cout << "-----------" << endl;
            cout << "enter product code: ";
            cin >> code;
            ans.erase(code); // удаление товара
            cout << "-----------" << endl;
            getline(cin, command);
            cout << "-> enter next command: ";
        }
        else if (command == "/value") {
            int code;
            cout << "-----------" << endl;
            cout << "enter product code: ";
            cin >> code;
            goods product = ans.value(code); // найденный товар по коду
            cout << "code: " << product.code << endl;
            cout << "title: " << product.title << " | cost: " << product.cost << endl;
            cout << "-----------" << endl;
            getline(cin, command);
            cout << "-> enter next command: ";
        }
        else if (command == "/output") {
            ans.output(); // вывод таблицы
            cout << "-> enter next command: ";
        }
        else if (command == "/end") {
            cout << "-----------" << endl;
            cout << "programm ended", exit(0);
        }
        else {
            cout << "-----------" << endl;
            cout << "wrong command" << endl;
            cout << "-----------" << endl;
            cout << "-> enter next command: ";
        }
    }
    return 0;
}
```

Тестирование

Тестирование программы приведено на рисунках 1, 2, 3.


Консоль отладки Microsoft Visual Studio
choose input way: keyboard/random
random
enter amount of random products to fill: 10
logging: table rehashed with size of 3
logging: table rehashed with size of 6
logging: table rehashed with size of 12
---- hash table ----
---- 1 product ----
code: 107716
title: random_0 | cost: 172
---- 2 product ----
code: 111811
title: random_1 | cost: 3577
---- 3 product ----
code: 125773
title: random_2 | cost: 12706
---- 4 product ----
code: 108617
title: random_3 | cost: 1940
---- 5 product ----
code: 119697
title: random_4 | cost: 1449
---- 6 product ----
code: 122230
title: random_5 | cost: 29511
---- 7 product ----
code: 109657
title: random_6 | cost: 13003
---- 8 product ----
code: 120659
title: random_7 | cost: 28087
---- 9 product ----
code: 130293
title: random_8 | cost: 4270
---- 10 product ----
code: 111222
title: random_9 | cost: 31844
----
list of commands

/insert - adding new product
/erase - deleting product
/value - searching value by key
/output - display hash table
/end - ending programm

Рисунок 1 - тестирование программы

```
-> enter next command: /erase
-----------
enter product code: 120659
-----------
-> enter next command: /insert
-----------
enter product code: 111210
enter product title: kolliziya
enter product cost: 10
-----------
-> enter next command: /value
-----------
enter product code: 111210
code: 111210
title: kolliziya | cost: 10
-----------
```

Рисунок 2 - тестирование программы

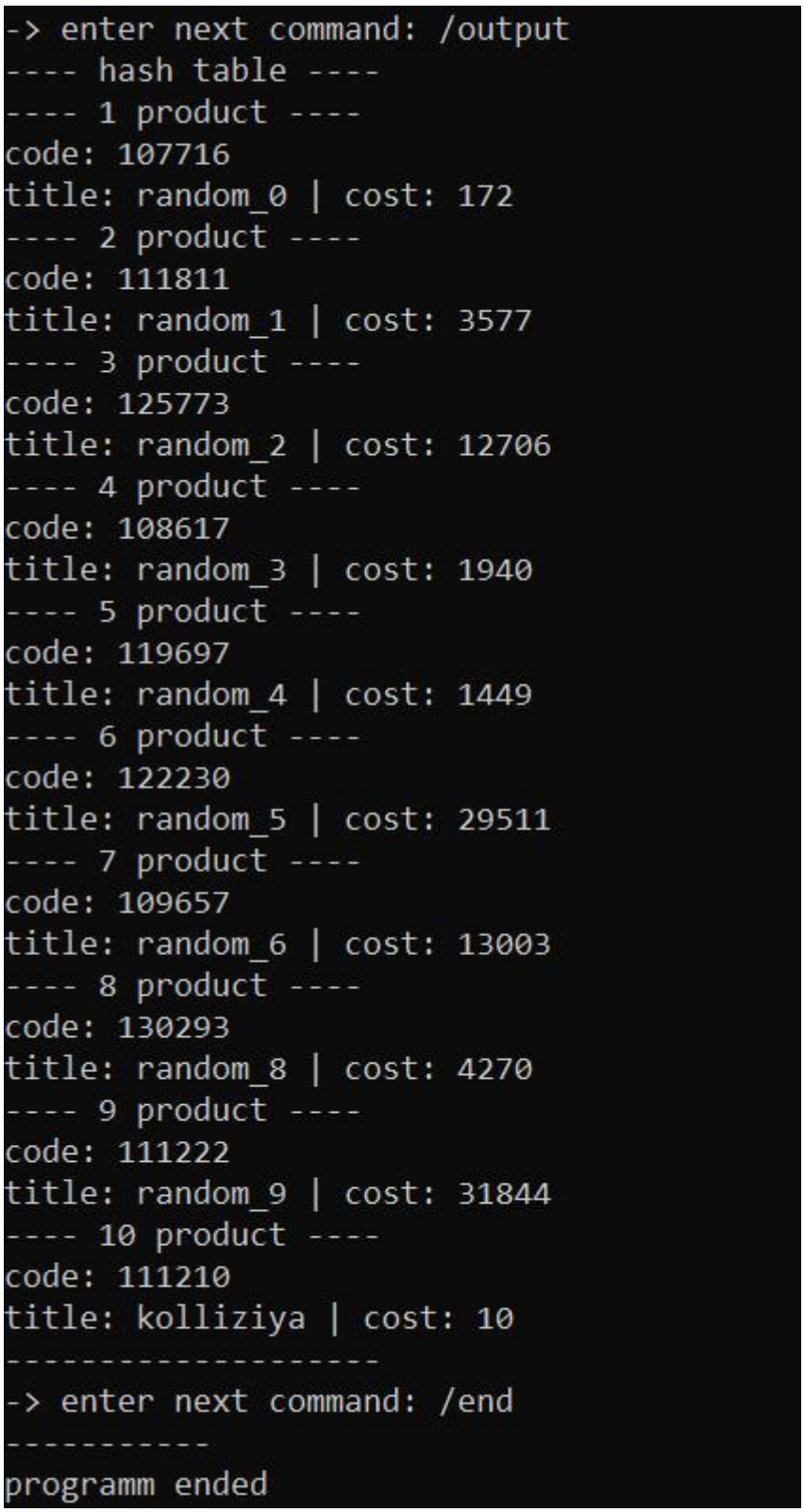


Рисунок 3 - тестирование программы

Таким образом, по результатам тестирования видно, что все базовые операции, а также механизмы разрешения коллизий, расширения массива и рехэширования всех добавленных записей после расширения отрабатывают корректно.

Вывод

В результате выполнения данной работы мной были освоены навыки хэширования и эффективного поиска элементов в множестве. В результате практического применения хэш-функций в реализации собственного динамического множества с хэш-таблицей мной был получен опыт использования хэширования на практике в целях повышения эффективности работы с элементами динамического множества, их поиска, добавления, а также удаления.