



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания №6.1

Тема: Быстрый доступ к данным с помощью хеш-таблиц

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент
Группа

Зуев Д.А.
ИКБО-68-23

Москва 2025

Цель работы: освоить приёмы хеширования и эффективного поиска элементов множества

ЗАДАНИЕ

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.

Персональный вариант:

№	Поиск	Структура записи
29	Открытая адресация (линейное пробирование)	Читательский абонемент: номер читательского - целое пятизначное число, ФИО, адрес

Математическая модель решения (описание алгоритма)

В данной задаче мы разрабатываем хеш-таблицу для организации прямого доступа к элементам динамического множества полезных данных. Хеш-таблица будет использоваться для хранения товаров, каждый из которых имеет уникальный код, название и цену. В этой модели мы применяем метод разрешения коллизий через квадратичное пробирование.

1. Структура данных

Элемент данных `Library_card`:

Каждый абонемент представляется структурой, `Library_card` содержащей:

- `code`: строка (пятиразрядное число) — уникальный код товара.
- `name`: строка — ФИО.
- `address`: строка — Адрес.

Хеш-таблица `hash_table`:

Хеш-таблица представлена вектором указателей на элементы `Library_card`.

Поля:

- `size`: текущее количество мест в таблице.
- `count`: текущее количество элементов в таблице.

2. Алгоритм работы хеш-таблицы

Хеширование:

Для вычисления индекса элемента используется хеш-функция:

$$h(key) = key \bmod size,$$

где `key` — это код карты, а `size` — размер таблицы.

Вставка элемента:

- 1) Проверяем, не превышает ли коэффициент загрузки (`count/size`) заданный порог (например, 0.7).
- 2) Если превышает, вызываем функцию рехеширования.
- 3) Вычисляем индекс с помощью хеш-функции.
- 4) Если по этому индексу уже находится элемент (коллизия), выполняем линейное пробирование:

$$index = (h(key) + c \cdot i) \bmod size$$

где `i` — номер текущей попытки разрешить коллизия, `c` — константа, равная 1.

Удаление элемента

- 1) Вычисляем индекс с помощью хеш-функции.
- 2) Используем линейное пробирование для поиска нужного элемента.
- 3) Если элемент найден, освобождаем память и устанавливаем значение по индексу в `nullptr`.

Поиск элемента

- 1) Вычисляем индекс с помощью хеш-функции.
- 2) Используем линейное пробирование для поиска элемента.
- 3) Если элемент найден, возвращаем указатель на него; если нет — возвращаем `nullptr`.

Рехеширование

- 1) Создаем новый вектор с увеличенным размером (например, удваиваем).
- 2) Перемещаем все элементы из старой таблицы в новую, пересчитывая индексы с использованием новой хеш-функции.

3. Интерфейс пользователя

Пользователь взаимодействует с программой через текстовый интерфейс, который позволяет выполнять следующие операции:

- 1) Автоматическое заполнение таблицы: Генерация случайных абонементов с уникальными кодами и добавление их в таблицу.
- 2) Ручное заполнение таблицы: Ввод пользователем данных о карте (код, ФИО, адрес).
- 3) Отображение содержимого таблицы: Вывод всех элементов, находящихся в таблице.
- 4) Поиск карты по коду: Поиск и вывод информации о карты по указанному коду.
- 5) Удаление карты по коду: Удаление элемента из таблицы по указанному коду.

Код программы

```

1957     const int INITIAL_SIZE = 7; // Начальный размер хеш-таблицы
1958     const double LOAD_FACTOR = 0.7; // Коэффициент загрузки для увеличения размера
1959
1960     struct Library_card {
1961         std::string code; // Код товара (шестиразрядное число)
1962         std::string name; // ФИО
1963         std::string address; // адрес
1964
1965         Library_card(std::string c, std::string n, std::string p) : code(c), name(n), address(p) {}
1966     };
1967
1968     class HashTable {
1969     private:
1970         std::vector<Library_card*> table;
1971         int size;
1972         int count;
1973
1974     public:
1975         int hashFunction(const std::string& key) {
1976             return std::stoi(key) % size; // Хеш-функция: остаток от деления
1977         }
1978
1979         void rehash() {
1980             std::vector<Library_card*> oldTable = table;
1981             size *= 2; // Увеличение размера таблицы
1982             table = std::vector<Library_card*>(size, nullptr);
1983             count = 0;
1984
1985             for (Library_card* card : oldTable) {
1986                 if (card != nullptr) {
1987                     insert(card->code, card->name, card->address);
1988                     delete card; // Освобождение памяти
1989                 }
1990             }
1991         }
1992     };

```

Рисунок 1 – инициализация констант, реализация структуры для записи данных, реализация класса хэш-таблицы (часть 1)

```

1992     public:
1993         HashTable() : size(INITIAL_SIZE), count(0) {
1994             table = std::vector<Library_card*>(size, nullptr);
1995         }
1996
1997         ~HashTable() {
1998             for (Library_card* product : table) {
1999                 delete product;
2000             }
2001         }
2002
2003         void insert(const std::string& code, const std::string& name, std::string address) {
2004             if ((double)count / size >= LOAD_FACTOR) {
2005                 rehash(); // Проверка на необходимость рехеширования
2006             }
2007
2008             int index = hashFunction(code);
2009             int i = 0;
2010             const int c = 1; // Константа для линейного пробирования
2011
2012             while (table[index] != nullptr) {
2013                 index = (index + c * i) % size; // Пробирование
2014                 i++;
2015             }
2016
2017             table[index] = new Library_card(code, name, address);
2018             count++;
2019         }
2020
2021         void remove(const std::string& code) {
2022             int index = hashFunction(code);
2023             int i = 0;
2024             const int c = 1;

```

Рисунок 2 – реализация класса хэш-таблицы (часть 2)

```

2038     Library_card* search(const std::string& code) {
2039         int index = hashFunction(code);
2040         int i = 0;
2041         int c = 1;
2042
2043         while (table[index] != nullptr) {
2044             if (table[index]->code == code) {
2045                 return table[index]; // Найден товар
2046             }
2047             index = (index + i * c) % size; // Пробирование
2048             i++;
2049         }
2050         return nullptr; // Товар не найден
2051     }
2052
2053     void display() {
2054         for (int i = 0; i < size; i++) {
2055             if (table[i] != nullptr) {
2056                 std::cout << "Индекс " << i << ": "
2057                     << "Код: " << table[i] -> code
2058                     << ", ФИО: " << table[i] -> name
2059                     << ", Адрес: " << table[i] -> address << "\n";
2060             }
2061             else {
2062                 std::cout << "Индекс " << i << ": пусто" << "\n";
2063             }
2064         }
2065     }
2066

```

Рисунок 3 – реализация класса хэш-таблицы (часть 3)

```

2067     void auto_Fill(int numProducts) {
2068         srand(static_cast<unsigned>(time(0))); // Инициализация генератора случайных чисел
2069
2070         for (int i = 0; i < numProducts; ++i) {
2071             std::string code = std::to_string(rand() % 1000000); // Генерация случайного кода
2072             while (code.length() < 6) { // Дополнение до шести разрядов
2073                 code.insert(0, "0");
2074             }
2075             insert(code, "ФИО " + code, "Ул." + std::to_string(rand() % 100 + 1)); // Генерация названия
2076         }
2077     }
2078
2079     void manual_Fill(int numProducts) {
2080         for (int i = 0; i < numProducts; ++i) {
2081             std::string code, name, address;
2082
2083             std::cout << "Введиет код товара (5 цифр): ";
2084             std::cin >> code;
2085
2086             while (code.length() != 5 || !isdigit(code[0])) {
2087                 std::cout << "Неправильный код. Пожалуйста, введите пятизначный код: ";
2088                 std::cin >> code;
2089             }
2090
2091             std::cout << "Введите ФИО: ";
2092             std::cin.ignore(); // Очистка буфера ввода
2093             std::getline(std::cin, name);
2094
2095             std::cout << "Введите Адрес: ";
2096             std::cin.ignore();

```

Рисунок 4 – реализация класса хэш-таблицы (часть 4)

```

2103
2104 int main() {
2105     setlocale(LC_ALL, "rus");
2106     HashTable ht;
2107     int choice;
2108     do {
2109         std::cout << "Меню:\n";
2110         std::cout << "1. Автоматическое заполнение\n";
2111         std::cout << "2. Ручное заполнение\n";
2112         std::cout << "3. Вывод таблицы\n";
2113         std::cout << "4. Поиск товара\n";
2114         std::cout << "5. Удаление товара\n";
2115         std::cout << "6. Выход\n";
2116         std::cout << "Выберите действие: ";
2117         std::cin >> choice;
2118         switch (choice) {
2119             case 1: {
2120                 std::cout << "Введите количество данных для автоматического заполнения: ";
2121                 int count;
2122                 std::cin >> count;
2123                 ht.auto_Fill(count); // Автоматическое заполнение таблицы
2124                 break;
2125             }
2126             case 2: {
2127                 int n;
2128                 std::cout << "Введите количество данных для ручного заполнения: ";
2129                 std::cin >> n;
2130                 ht.manual_Fill(n); // Ручное заполнение таблицы
2131                 break;
2132             }
2133             case 3: {
2134                 std::cout << "\nТекущая таблица:\n";
2135                 ht.display();
2136                 break;
2137             }
2138             case 4: {
2139                 std::string code;
2140                 std::cout << "Введите код карты для поиска: ";
2141                 std::cin >> code;
2142                 Library_card* found = ht.search(code);
2143                 if (found != nullptr) {
2144                     std::cout << "Найдено: " << found->name << ", Цена: " << found->address << "\n";
2145                 }
2146                 else {
2147                     std::cout << "Данные не найдены" << "\n";
2148                 }
2149             }
2150             default: {
2151                 std::cout << "Неверный выбор действия\n";
2152             }
2153         }
2154     } while (choice != 6);
2155     return 0;
2156 }

```

Рисунок 5 – реализация основной функции программы с «пультом управления»

Результаты тестирования

Проведем тестирование программы с коллизиями:

```

Индекс 0: пусто
Индекс 1: Код: 11111, ФИО: Oleg, Адрес: ushkina 4
Индекс 2: Код: 22221, ФИО: Sam, Адрес: enina 7

```

Рисунок 6 – тестирование преодоления коллизий

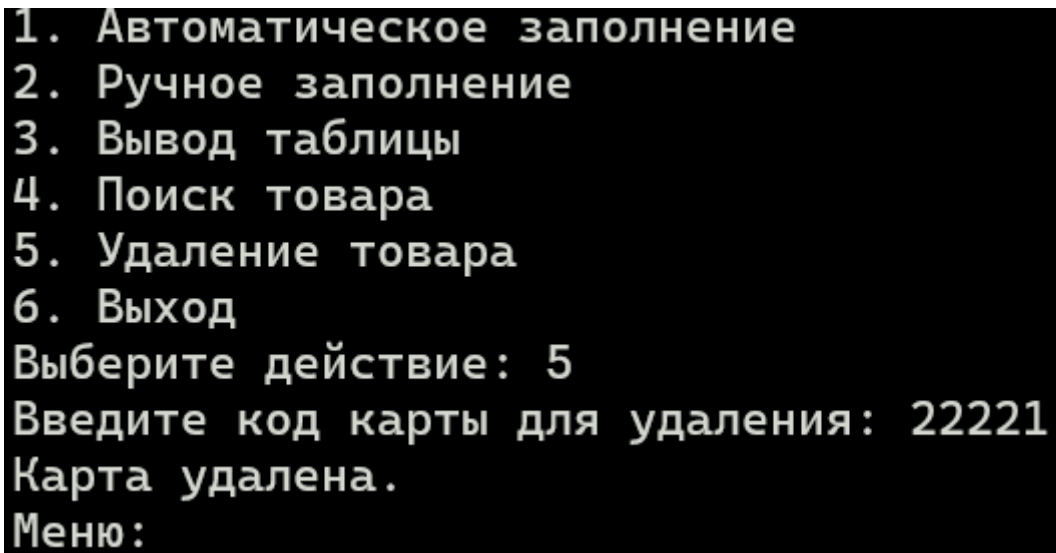
Тестирование показало, что преодоление коллизий с помощью Линейного пробирования работает корректно.

```

Выберите действие: 4
Введите код карты для поиска: 11111
Найдено: Oleg, Цена: ushkina 4

```

Рисунок 7 – тестирование функции поиска



```
1. Автоматическое заполнение
2. Ручное заполнение
3. Вывод таблицы
4. Поиск товара
5. Удаление товара
6. Выход
Выберите действие: 5
Введите код карты для удаления: 22221
Карта удалена.
Меню:
```

Рисунок 8 – тестирование функции удаления

Тестирование показало, что алгоритм работает корректно.

ВЫВОД

В ходе выполнения задания была разработана программа, использующая хеш-таблицу для доступа к элементам динамического множества данных. Основной целью было освоение методов хеширования и эффективного поиска. Программа включает операции вставки, удаления, поиска и обработки коллизий с помощью квадратичного пробирования. Реализовано динамическое изменение размера таблицы и удобный для пользователя текстовый интерфейс. Тестирование подтвердило корректность работы приложения. Разработка углубила понимание принципов хеширования и их применения в программировании.