МИНОБРНАУКИ РОССИИ

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)

ОТЧЁТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ

«Алгоритмы поиска»

по дисциплине «Структуры и алгоритмы обработки данных (часть 2/2)»

Выполнил студент группы ИКБО-41-23 Попов А.В.

Принял Рысин М.Л.
Ассистент

Практические работы выполнены «___»_____2024 г. (подпись студента)

«Зачтено» «___»_____2024 г. (подпись преподавателя)

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Освоить приёмы хеширования и эффективного поиска элементов множества. Разработать приложение, которое использует хеш-таблицу для организации прямого доступа к элементам динамического множества полезных данных. Реализовать текстовый командный интерфейс пользователя для возможности вызова методов в любой произвольной последовательности.

2 ЗАДАНИЕ

2.1 Формулировка задачи

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

Приложение должно содержать класс с базовыми операциями: вставки, удаления, поиска по ключу, вывода. Включить в класс массив полезных данных и хеш-таблицу. Хеш-функция подберается самостоятельно, с использованием правил выбора функции.

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

Предусмотреть автоматическое заполнение таблицы 5-7 записями.

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

Провести полное тестирование программы (все базовые операции, изменение размера и рехеширование).

Персональный вариант №11:

Метод хеширования (тип последовательностей проб): открытая адресация (двойное хеширование).

Структура элемента множества: номер телефона – последовательность 10 символов, адрес.

2.2 Математическая модель решения

Хеш-таблица использует две хеш-функции:

1. Первая хеш-функция:

$$h_1(k) = (k \bmod p_1) \bmod m$$

где k – ключ (номер телефона), p_1 – простое число (по умолчанию 31), m – текущий размер таблицы.

2. Вторая хеш-функция:

$$h_2(k) = (k \bmod p_2) \bmod m$$

где k – ключ (номер телефона), p_2 – другое простое число (по умолчанию 17), m – текущий размер таблицы.

Эти функции помогают определить начальную позицию ключа в таблице, а также шаг для разрешения коллизий. Коллизия разрешается методом двойного хеширования: при столкновении (когда ячейка уже занята) следующая проверяемая позиция вычисляется по формуле:

$$index_{следующий} = (index_{текущий} + h_2(k)) \bmod m$$

Когда нагрузка таблицы (отношение числа элементов к размеру таблицы) превышает 0.5, таблица удваивается в размере. Для этого выполняются следующие действия:

1. Создается новая таблица, в два раза больше предыдущей.
2. Все существующие элементы из старой таблицы перехешируются в новую. Для этого каждый ключ обрабатывается заново, с использованием обновленных значений размера таблицы.

Интерфейс позволяет пользователю управлять хеш-таблицей с помощью команд: добавления, удаления, поиска и отображения элементов.

Каждая команда обрабатывается соответствующей функцией. При необходимости интерфейс вызывает операции рехеширования или сообщает об ошибках, например, если пользователь пытается найти или удалить несуществующий элемент.

Для хранения номера телефона в программе используется тип данных ***long long int***. Этот выбор обусловлен следующими соображениями:

- Номер телефона состоит из 10 цифр. Например, номер вида 9154318262 представляет собой числовое значение, которое занимает 10 разрядов. Тип ***long long int*** гарантирует, что такие числа могут быть корректно представлены без риска переполнения.
- Использование числового представления позволяет легко вычислять хеш-функции. В данном случае операции вычисления остатка от деления и другие арифметические операции с номерами телефонов выполняются быстро и эффективно.

Если бы номера телефонов хранились как строки (***string***), обработка данных потребовала бы дополнительных вычислительных затрат. Таким образом, выбор ***long long int*** для хранения телефонных номеров делает программу более простой, эффективной и безопасной с точки зрения корректного представления данных.

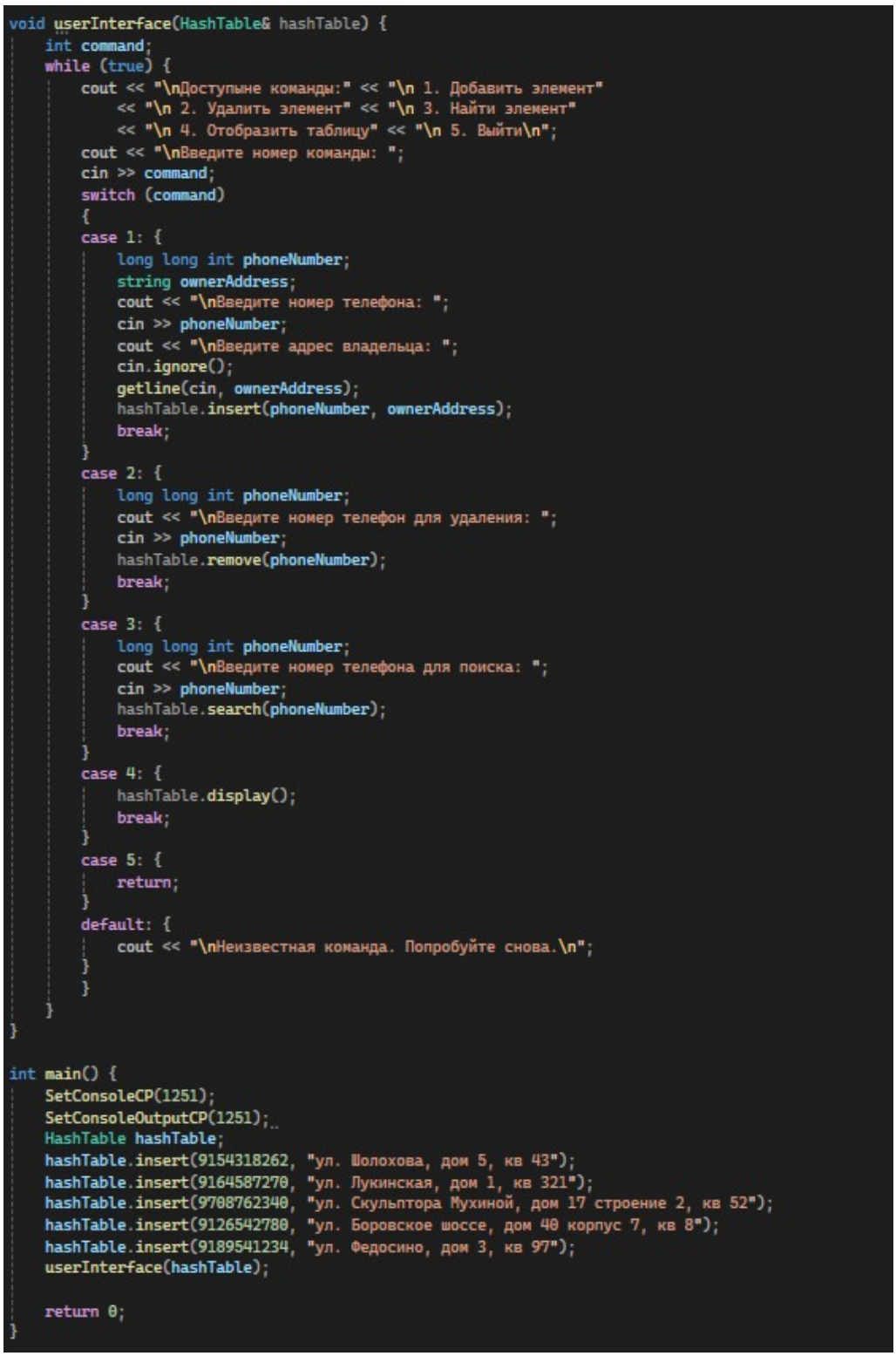
2.3 Реализация задачи

```
#include <iostream>
#include <vector>
#include <string>
#include <Windows.h>

using namespace std;

struct Data {
    long long int phoneNumber;
    string ownerAddress;
};

class HashTable {
private:
    vector<Data*> table;
    int size;
    int count;

    int hash(long long int key, int prime = 31) const {
        return (key % prime) % size;
    }

    int hash2(long long int key, int prime = 17) const {
        return (key % prime) % size;
    }

    void rehash() {
        cout << "\nРасширение размера таблицы и рехеширование.\n";
        vector<Data*> old_table = table;
        size *= 2;
        table.clear();
        table.resize(size, nullptr);
        count = 0;
        for (auto element : old_table) {
            if (element != nullptr) {
                insert(element->phoneNumber, element->ownerAddress, false);
            }
            delete element;
        }
    }
```

Рисунок 1 — Структура множества полезных данных, класс хеш-таблицы и описанные в нём методы хеширования и рехеширования.

```
public:
    HashTable(int initial_size = 8) : size(initial_size), count(0) {
        table.resize(size, nullptr);
    }

    ~HashTable() {
        for (auto element : table) {
            delete element;
        }
    }

    void insert(long long int phoneNumber, const string& ownerAddress, bool message_flag=true) {
        if (count > size / 2) {
            rehash();
        }

        int index = hash(phoneNumber);
        int step = hash2(phoneNumber);
        while (table[index] != nullptr) {
            index = (index + step) % size;
        }

        table[index] = new Data{ phoneNumber, ownerAddress };
        count++;
        if (message_flag) {
            cout << "\nЭлемент с номером телефона " << phoneNumber << " добавлен.\n";
        }
    }

    void search(long long int phoneNumber) const {
        int index = hash(phoneNumber);
        int step = hash2(phoneNumber);
        for (int i = 0; i < size; i++) {
            if (table[index] != nullptr && table[index]->phoneNumber == phoneNumber) {
                cout << "\nНайден элемент: номер телефона = " << phoneNumber << ", адрес владельца = " << table[index]->ownerAddress << "\n";
                return;
            }
            index = (index + step) % size;
        }
        cout << "\nНомер телефона " << phoneNumber << " не найден.\n";
    }

    void remove(long long int phoneNumber) {
        int index = hash(phoneNumber);
        int step = hash2(phoneNumber);
        for (int i = 0; i < size; i++) {
            if (table[index] != nullptr && table[index]->phoneNumber == phoneNumber) {
                delete table[index];
                table[index] = nullptr;
                count--;
                cout << "\nЭлемент с номером телефона " << phoneNumber << " удален.\n";
                return;
            }
            index = (index + step) % size;
        }
        cout << "\nОшибка: номер телефона " << phoneNumber << " не найден.\n";
    }

    void display() const {
        cout << "\nСодержимое хеш-таблицы:" << endl;
        for (int i = 0; i < size; ++i) {
            if (table[i] != nullptr) {
                cout << "\nЯчейка " << i << ": номер телефона = " << table[i]->phoneNumber << ", адрес владельца = " << table[i]->ownerAddress << endl;
            }
            else {
                cout << "\nЯчейка " << i << ": пусто" << endl;
            }
        }
        cout << endl;
    }
};
```

Рисунок 2 — Конструктор и декструктор таблицы, методы добавления, поиска, удаления и вывода хеш-таблицы



```
void userInterface(HashTable& hashTable) {
    int command;
    while (true) {
        cout << "\nДоступыне команды:" << "\n 1. Добавить элемент"
            << "\n 2. Удалить элемент" << "\n 3. Найти элемент"
            << "\n 4. Отобразить таблицу" << "\n 5. Выйти\n";
        cout << "\nВведите номер команды: ";
        cin >> command;
        switch (command)
        {
        case 1: {
            long long int phoneNumber;
            string ownerAddress;
            cout << "\nВведите номер телефона: ";
            cin >> phoneNumber;
            cout << "\nВведите адрес владельца: ";
            cin.ignore();
            getline(cin, ownerAddress);
            hashTable.insert(phoneNumber, ownerAddress);
            break;
        }
        case 2: {
            long long int phoneNumber;
            cout << "\nВведите номер телефон для удаления: ";
            cin >> phoneNumber;
            hashTable.remove(phoneNumber);
            break;
        }
        case 3: {
            long long int phoneNumber;
            cout << "\nВведите номер телефона для поиска: ";
            cin >> phoneNumber;
            hashTable.search(phoneNumber);
            break;
        }
        case 4: {
            hashTable.display();
            break;
        }
        case 5: {
            return;
        }
        default: {
            cout << "\nНеизвестная команда. Попробуйте снова.\n";
        }
        }
    }
}

int main() {
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);..
    HashTable hashTable;
    hashTable.insert(9154318262, "ул. Шолохова, дом 5, кв 43");
    hashTable.insert(9164587270, "ул. Лукинская, дом 1, кв 321");
    hashTable.insert(9708762340, "ул. Скульптора Мухиной, дом 17 строение 2, кв 52");
    hashTable.insert(9126542780, "ул. Боровское шоссе, дом 40 корпус 7, кв 8");
    hashTable.insert(9189541234, "ул. Федосино, дом 3, кв 97");
    userInterface(hashTable);

    return 0;
}
```

Рисунок 3 — функция, реализующая пользовательский интерфейс, основная функция программы и автоматическое заполнение таблицы записями

2.4 Результаты тестирования

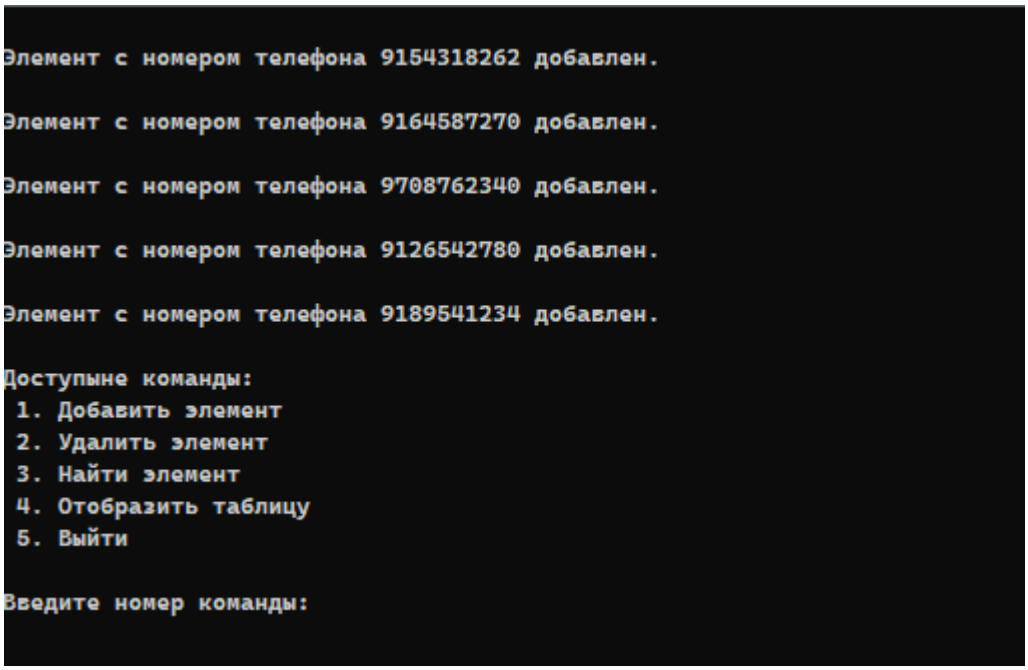


Рисунок 4 — Начальное состояние после запуска

После запуска программы мы имеем на выбор 5 команд (Рисунок 4). Для их выполнения следует ввести номер необходимой команды и заполнить её аргументы.

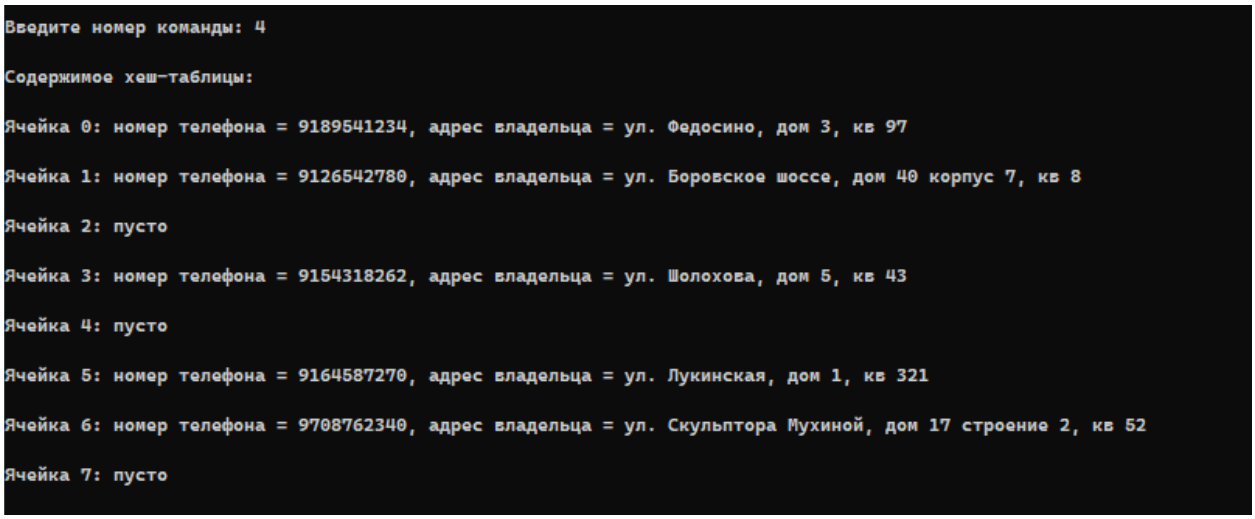


Рисунок 5 — Результат вывода хеш-таблицы

Как мы можем заметить (Рисунок 5), выводятся все ячейки текущей хеш-таблицы, в формате «Ячейка N: множество полезных данных», где N –

хеш, полученный в результате применения алгоритма хеширования на номере телефона (который является ключевым значением).

Так как хеш-таблица уже заполнена больше, чем на половину, 5 из 8 элементов, то после добавления ещё одного таблица будет рехеширована и новое значения появится в ней (Рисунок 6).

```
Введите номер команды: 1

Введите номер телефона: 9451924532

Введите адрес владельца: ул. Покрышкина, дом 10, кв 482

Расширение размера таблицы и рехеширование.

Элемент с номером телефона 9451924532 добавлен.

Доступные команды:
 1. Добавить элемент
 2. Удалить элемент
 3. Найти элемент
 4. Отобразить таблицу
 5. Выйти

Введите номер команды: 4

Содержимое хеш-таблицы:

Ячейка 0: номер телефона = 9189541234, адрес владельца = ул. Федосино, дом 3, кв 97

Ячейка 1: пусто

Ячейка 2: пусто

Ячейка 3: номер телефона = 9154318262, адрес владельца = ул. Шолохова, дом 5, кв 43

Ячейка 4: пусто

Ячейка 5: пусто

Ячейка 6: пусто

Ячейка 7: пусто

Ячейка 8: пусто

Ячейка 9: номер телефона = 9126542780, адрес владельца = ул. Боровское шоссе, дом 40 корпус 7, кв 8

Ячейка 10: пусто

Ячейка 11: номер телефона = 9451924532, адрес владельца = ул. Покрышкина, дом 10, кв 482

Ячейка 12: пусто

Ячейка 13: номер телефона = 9164587270, адрес владельца = ул. Лукинская, дом 1, кв 321

Ячейка 14: номер телефона = 97087623340, адрес владельца = ул. Скульптора Мухиной, дом 17 строение 2, кв 52

Ячейка 15: пусто
```

Рисунок 6 — Добавление нового элемента и рехеширование таблицы

```
Введите номер команды: 2

Введите номер телефон для удаления: 9154318262

Элемент с номером телефона 9154318262 удален.

Доступыне команды:
 1. Добавить элемент
 2. Удалить элемент
 3. Найти элемент
 4. Отобразить таблицу
 5. Выйти

Введите номер команды: 4

Содержимое хеш-таблицы:

Ячейка 0: номер телефона = 9189541234, адрес владельца = ул. Федосино, дом 3, кв 97

Ячейка 1: пусто

Ячейка 2: пусто

Ячейка 3: пусто

Ячейка 4: пусто

Ячейка 5: пусто

Ячейка 6: пусто

Ячейка 7: пусто

Ячейка 8: пусто

Ячейка 9: номер телефона = 9126542780, адрес владельца = ул. Боровское шоссе, дом 40 корпус 7, кв 8

Ячейка 10: пусто

Ячейка 11: номер телефона = 9451924532, адрес владельца = ул. Покрышкина, дом 10, кв 482

Ячейка 12: пусто

Ячейка 13: номер телефона = 9164587270, адрес владельца = ул. Лукинская, дом 1, кв 321

Ячейка 14: номер телефона = 9708762340, адрес владельца = ул. Скульптора Мухиной, дом 17 строение 2, кв 52

Ячейка 15: пусто
```

Рисунок 7 — Удаление элемента

```
Введите номер команды: 3

Введите номер телефона для поиска: 9164587270

Найден элемент: номер телефона = 9164587270, адрес владельца = ул. Лукинская, дом 1, кв 321
```

Рисунок 8 — Поиск элемента по номеру (ключу)

3 ВЫВОД

В ходе выполнения работы было разработано приложение, использующее хеш-таблицу для организации прямого доступа к данным. В приложении реализованы основные операции: вставка, удаление, поиск по ключу и вывод. Хеш-таблица работает на массиве, где в качестве ключа используется номер телефона.

Для работы с коллизиями была реализована рехешировка и расширение таблицы, когда она заполняется. Также добавлен текстовый командный интерфейс для удобного взаимодействия с пользователем.

Все основные операции были протестированы и успешно выполнены. Программа корректно работает при различных тестах, включая рехеширование и изменение размера таблицы.

4 Ответы на вопросы

1. Хеширование — процесс преобразования данных произвольной длины в фиксированный хеш с помощью хеш-функций. Применяется в хранении данных (хеш-таблицы), криптографии, сетевых технологиях, поиске дубликатов и анализе больших данных.

2. Основные свойства хеш-функции: определённость, высокая скорость, односторонность, равномерность распределения, стойкость к коллизиям.

3. Алгоритмы хеширования основаны на арифметических и побитовых операциях, использовании простых чисел и комбинировании блоков данных.

4. Константная вычислительная сложность (O(1)) означает, что время выполнения операции не зависит от размера данных. Хеширование позволяет реализовать её за счёт прямого доступа к данным в хеш-таблице.

5. Коллизия — ситуация, когда разные данные имеют одинаковый хеш. Методы устранения: цепное хеширование, открытая адресация, двойное хеширование, рехеширование.

6. Цепное хеширование — метод разрешения коллизий, при котором элементы с одинаковым хешем хранятся в списке. Проблема: увеличение сложности операций из-за длинных цепочек при частых коллизиях.

7. Рехеширование — изменение размера хеш-таблицы и перераспределение элементов при переполнении или высоком коэффициенте заполнения.

8. Открытая адресация — способ разрешения коллизий, при котором поиск свободной ячейки осуществляется внутри самой таблицы.

9. Наиболее распространённые схемы последовательности проб: линейное, квадратичное пробирование и двойное хеширование.