



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 №2

Тема: Хеширование и организация быстрого поиска данных

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент

Хан А.А.

группа

ИКБО-04-20

Москва 2021

Оглавление

Тема	3
Цель.....	3
Ответы на вопросы.....	3
Персональный вариант	5
Постановка задачи.....	5
Пояснение к коду	7
Код приложения	9
Скриншот результатов выполнения операции с хеш-таблицей.....	13
Скриншоты содержания файла и хеш-таблицы во время работы	15
Выводы	16
Список информационных источников.....	16

Тема

Хеширование и организация быстрого поиска данных

Цель

Получить навыки по разработке хеш-таблиц и их применении

Ответы на вопросы

1. Расскажите о назначении хеш-функции.

Хеширование (хеш-функция) создаёт отображение (соответствие) множества ключей в множество индексов массива (множество целых чисел).

2. Что такое коллизия?

Коллизия – это ситуация, когда для двух разных ключей хеш-функция создает одинаковый индекс.

3. Что такое «открытый адрес» по отношению к хеш-таблице?

Для начала разберемся, что такое хеш-таблица.

Хеш-таблица – это структура, позволяющая хранить пары «ключ» – «хеш-код» (индекс). Она может хранить как сами элементы данных (записи) исходного динамического множества, так и ссылки на эти данные.

Организация таблицы с открытым адресом – это другой метод разрешения коллизий.

Открытый адрес – это свободная ячейка таблицы, а закрытый адрес – это занятая ячейка.

Суть метода: значение вставляется непосредственно в таблицу в открытую ячейку (массив из записей с ключами).

4. Как в хеш-таблице с открытым адресом реализуется коллизия?

В случае появления коллизии, осуществляется поиск свободной ячейки, путем линейного опробования, следующих ячеек (индекс увеличивается на единицу), пока не будет найдена свободная ячейка. Эффективность данного подхода падает, когда коэффициент нагрузки приближается к единице.

5. Какая проблема, может возникнуть после удаления элемента из хеш-таблицы с открытым адресом и как ее устранить?

Выполнение операции удаления записи из хеш-таблицы с открытым адресом имеет свои трудности и связано это с тем, что при удалении нельзя делать ячейку открытой, так как другие ключи при разрешении коллизий были вставлены после удаляемой и поиск свободной выполнялся при закрытой ячейке.

При выполнении операции удаления сначала осуществляется поиск записи с заданным ключом в таблице, а затем, в случае удачного поиска, и выполняется удаление записи из таблицы, т.е. ячейка должна стать открытой. Выполненная операция удаления, в таком виде, как мы ее только что рассмотрели, породила проблему поиска.

Признак открытого адреса удаленного элемента находится в значении true – т.е. ячейка свободна и процесс поиска завершается, но элемент не найден, хотя запись с таким ключом присутствует в таблице.

Чтобы устранить эту проблему операции удаления, удобно в таблицу ввести еще одно поле – признак удаленной записи. Это поле, так же, как и поле признак открытого адреса, может быть логического типа и содержать true, если запись удалена и false, если не удалена.

6. Что определяет коэффициент нагрузки в хеш-таблице?

Кол-во записей в таблице / максимальный размер таблицы.

Обычно это значение не должно превышать 0,75.

7. Что такое «первичный кластер» в таблице с открытым адресом?

Первичный кластер(или первичная кластеризация) – это явление, которое происходит, когда обработчик коллизий создает условие роста кластера. Обработчик коллизий со смещением 1 из таблицы с открытым адресом способствует первичной кластеризации. Первичная кластеризация порождает длинные пути.

8. Как реализуется двойное хеширование?

Помимо хеширования ключа, идёт хеширования смещения для устранения первичной кластеризации. Все элементы хранятся непосредственно в хеш-таблице, без использования связанных списков. В отличие от хеширования с цепочками, при использовании этого метода может возникнуть ситуация, когда хеш-таблица окажется полностью заполненной, следовательно, будет невозможно добавлять в неё новые элементы. Так что при возникновении такой ситуации решением может быть динамическое увеличение размера хеш-таблицы, с одновременной её рехешированием.

Персональный вариант

Персональный вариант: 28%17+1=12

№	Метод хеширования (способ реализации коллизий)	Структура записи файла (ключ – подчеркнутое поле)
12	Цепное хеширование	Регистрация малого предприятия: номер лицензии, название, учредитель

Постановка задачи

Разработайте приложение, которое использует хеш-таблицу для организации прямого доступа к записям файла, структура записи которого приведена в варианте.

Дано.

Хеш-таблица для реализации коллизий по методу Цепное хеширование.

Файл двоичный с записями фиксированной длины.

Структура записи файла слово, количество вхождений слова в текст

Результат.

Приложение, выполняющее операции

Управление хеш-таблицей: вставить ключ в таблицу, удалить ключ из таблицы, найти ключ в таблице, рехешировать таблицу

Управление файлом:

- 1) посредством хеш-таблицы: считать запись из файла при поиске записи
- 2) добавить запись в файл, удалить запись из файла, прочитав запись файла по заданному номеру записи.

2. Подход к решению

- 1) Хеш-таблица — класс
- 2) Структура элемента цепной хеш-таблицы: указатель на вершину списка (цепочки)
- 3) Структура элемента линейного однонаправленного списка: ключ, ссылка на запись в файле (это адрес места размещения записи/порядковый номер записи в файле), ссылка на следующий элемент списка. Новый элемент вставляется в начало списка.
- 4) Методы хеш-таблицы: вставить ключ в таблицу, удалить ключ из таблицы, найти ключ в таблице, решить таблицу, вывести хештаблицу в консоль.
- 5) Файл двоичный из записей фиксированного размера.

Структура записи файла:

слово — текст размером 50 символов,

количество вхождений — целое положительное число

признак логического удаления — логического типа

Операции по управлению файлом:

- добавить запись в файл: занести запись в файл, вставить сведения о записи в хеш-таблицу
- удалить запись из файла: удалить из хеш-таблицы и из файла
- прочитав запись из указанной позиции (номеру/смещению)

3. Алгоритмы операций на псевдокоде:

- вставка в таблицу
- поиск записи по ключу в таблице и возвращение ссылки на запись в файле
- удаление элемента из хеш-таблицы
- вставка записи по ключу

Пояснение к коду

Хеш-таблица – это структура данных, в которой все элементы хранятся в виде пары ключ-значение, где:

- Ключ – уникальное число, которое используется для индексации значений;
- Значение – это данные, которые с этим ключом связаны.

КЛЮЧ	ДАННЫЕ
------	--------

В хеш-таблице обработка новых индексов производится при помощи ключей. А элементы, связанные с этим ключом, сохраняются в индексе. Весь этот процесс называется хешированием.

В самом начале, я использовала шаблоны “template”, чтобы упростить себе в дальнейшем задачу, благодаря им создаются локальные версии шаблонированной функции для определенного набора параметров и типов данных.

Создала классы и конструкторы для Hashnode и Hashmap. Создала массив с хеш-элементами. Выделила начальную емкость для хеш-массива. Произвела инициализацию элементов и обозначила фиктивный узел. Можем наблюдать, что в коде одними из основных действий было: функция добавления пары ключ-значение и хеш-функция для поиска индекса по заданному ключу. Благодаря которым, в файл, который изначально был пустым, можно добавлять произвольное количество записей, удалять и добавлять новые данные. Основные моменты можно также наблюдать в сопровождающих код комментариях.

1) `template<typename K, typename V>`

Шаблон, для создания локальных версий функций.

2) `class HashMap`

Класс, в котором будет храниться наш массив с элементами, размер и узел.

3) `int hashCode(K key)`

Реализация поиска хэш-функции для поиска индекса/ключа.

4) `void insertNode(K key, V String)`

Функция добавления пары ключ-значение.

5) `int hashIndex = hashCode(key);`

Применить хэш-функцию для поиска индекса по заданному ключу.

6) `if(arr[hashIndex] == NULL || arr[hashIndex]->key == -1)`

Во время того, когда мы находим узел, мы также находим ему свободное место. Если будет вставлен новый узел, мы соответственно увеличиваем текущий размер. Когда наш узел с заданным ключом будет найден, мы вставляем фиктивный узел для дальнейшего использования и уменьшаем размер.

7) `V deleteNode(int key)`

Функция для удаления пары ключ-значение.

8) `void display()`

Функция для отображения сохраненных пар ключ-значение.

9) `h->deleteNode(123);`

После того, как мы заполнили нашу таблицу и файл. Мы можем удалить запись.

```
10)    h->insertNode(43,"Проектория Васильев Иван");
```

Добавить запись.

```
11)    cout<<h->get(43)<<endl;
```

А также найти строку по ключу.

Код приложения

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include<chrono>
#include <string>
#include <queue>
#include <vector>
#include <list>
using namespace std;

//шаблон для универсального типа
template<typename K, typename V>

//Hashnode класс
class HashNode
{
public:
    V value;
    K key;

    //Конструктор hashnode
    HashNode(K key, V value)
    {
        this->value = value;    this->key = key;
    }
};

//шаблон для универсального типа
template<typename K, typename V>

//Наш собственный HashMap класс
class HashMap
{
    //массив хэш-элементов
    HashNode<K, V>** arr;
    int capacity;    //текущий размер
    int size;    //фиктивный узел
    HashNode<K, V>* dummy;
public:    HashMap()
{
    //начальная емкость хэш-массива (начальная емкость)
    capacity = 20;
    size = 0;
```

```

arr = new HashNode<K, V>*[capacity];

//Инициализируем все элементы массива как NULL
for (int i = 0;
    i < capacity; i++)
    arr[i] = NULL;

//фиктивный узел со значением и ключом -1
dummy = new HashNode<K, V>(-1, "");
}

// Это реализует хэш-функцию для поиска индекса
// для ключа
int hashCode(K key)
{
    return key % capacity;
}

//Функция добавления пары ключ-значение
void insertNode(K key, V String)
{
    HashNode<K, V>* temp = new HashNode<K, V>(key, String);

    // Применить хэш-функцию для поиска индекса по заданному ключу
    int hashIndex = hashCode(key);

    //находим следующее свободное место
    while (arr[hashIndex] != NULL && arr[hashIndex]->key != key
        && arr[hashIndex]->key != -1)
    {
        hashIndex++;
        hashIndex %= capacity;
    }

    //если будет вставлен новые узел, увеличьте текущий размер
    if (arr[hashIndex] == NULL || arr[hashIndex]->key == -1)
        size++;
    arr[hashIndex] = temp;
}

//функция для удаления пары ключ-значение
V deleteNode(int key)
{
    // Применить хэш-функцию для поиска индекса по заданному ключу
    int hashIndex = hashCode(key);

    //находим узел с заданным ключом
    while (arr[hashIndex] != NULL)
    {
        //если узел найден
        if (arr[hashIndex]->key == key)
        {
            HashNode<K, V>* temp = arr[hashIndex];

            //Вставляем фиктивный узел для дальнейшего использования
            arr[hashIndex] = dummy;

            // Уменьшаем размер
            size--;
            return temp->value;
        }
        hashIndex++;
        hashIndex %= capacity;
    }

    //если не найдено, верните значение null
}

```

```

        return NULL;
    }

    //функция поиска значения для заданного ключа
    V get(int key)
    {
        // Применить хэш-функцию для поиска индекса по заданному ключу
        int hashIndex = hashCode(key);
        int counter = 0;
        //нахождение узла с заданным ключом
        while (arr[hashIndex] != NULL)
        {
            cout << "find key!";
            int counter = 0;
            if (counter++ > capacity) //чтобы избежать бесконечного цикла
                return NULL;
            //если узел найден, верните его значение
            if (arr[hashIndex]->key == key)
                return arr[hashIndex]->value;
            hashIndex++;
            hashIndex %= capacity;
        }

        //если не найдено, верните значение null
        return NULL;
    }

    //Возвращаем текущий размер
    int sizeofMap()
    {
        return size;
    }

    //Возвращаем истину, если размер равен 0
    bool isEmpty() {
        return size == 0;
    }

    //функция для отображения сохраненных пар ключ-значение
    void display() {
        for (int i = 0; i < capacity; i++)
        {
            if (arr[i] != NULL && arr[i]->key != -1)
                cout << "key = " << arr[i]->key << " value = " << arr[i]->value <<
endl;

            /*cout<< "key = " << arr[i]->key
               <<" value = "<< arr[i]->value << endl;*/
        }
        ofstream out("1.txt");
        if (out.is_open())
        {
            for (int i = 0; i < capacity; i++)
                if (arr[i] != NULL && arr[i]->key != -1)
                    out << arr[i]->key << " " << arr[i]->value << endl;
        }
        out.close();
    }

    //проверка на переполненность
    bool isOverflow() {
        return ((double)size / capacity) > 0.75 ? true : false;
    }
}

```

```

int hash(int code) {
    return code % this->capacity;
}
//рехэширование таблицы
void rehash() {
    this->size = 0;
    HashNode<K, V>;
    this->capacity *= 2;
    arr.clear();
    arr.resize(this->capacity);
    for (int i = 0; i < this->capacity / 2; i++) {
        while (!arr[i].empty()) {
            this->push(arr[i].front());
            arr[i].pop_front();
        }
    }
    arr.clear();
}
};

```

//Метод драйвера для проверки класса карты

```

int main()
{
    setlocale(LC_ALL, "rus");
    HashMap<int, string>* h = new HashMap<int, string>;
    // заполнить хеш-таблицу из файла
    ifstream f("1.txt");
    string str;
    if (f.is_open()) {
        while (getline(f, str))
        {
            cout << str << endl;

            string s_num = "";
            int i = 0;
            while (str[i] >= '1' && str[i] <= '9')
            {
                s_num += str[i];
                i++;
            }
            int number;
            number = std::atoi(s_num.c_str());
            h->insertNode(number, str.substr(i + 1, str.size()));
        }
    }
    f.close();
    cout << endl << "Состояние таблицы до удаления записи" << endl;
    h->display();

    //удалить запись из таблицы и файла

    h->deleteNode(123);

    cout << endl << "Измерение времени для вставки первого элемента" << endl;
    //добавить запись в таблицу и файл
    h->insertNode(43, "Проектория Васильев Иван");
    h->display();
    auto begin = chrono::high_resolution_clock::now();

    //Вызов операции поиска записи
    auto end = chrono::steady_clock::now();
}

```

```

chrono::duration<double, std::milli> elapsed_ms = end - begin;

cout << "Time: " << elapsed_ms.count() << "ms" << endl << endl;

cout << endl << "Измерение времени для вставки последнего элемента" << endl;

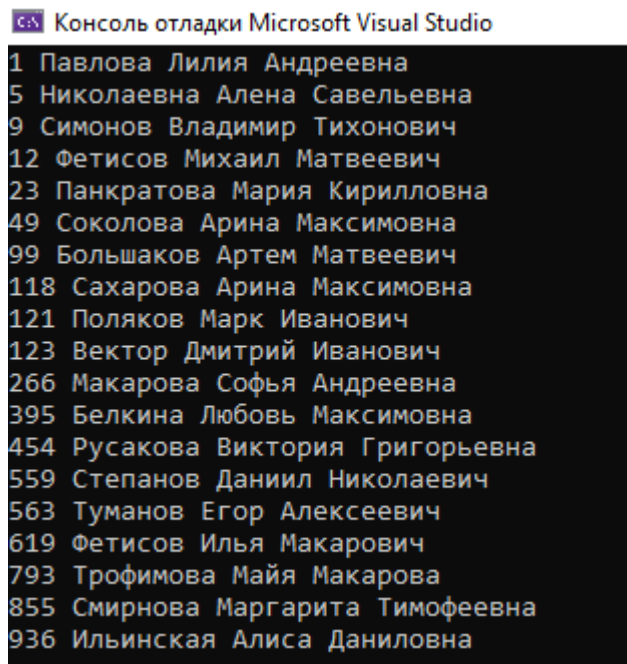
begin = chrono::high_resolution_clock::now();
h->insertNode(919, "Короткова Виктория Федоровна");
h->display();

//Вызов операции поиска записи
end = chrono::steady_clock::now();
elapsed_ms = end - begin;
cout << "Time: " << elapsed_ms.count() << "ms" << endl << endl;

//найти в таблице по ключу строку
cout << h->get(43) << endl;
return 0;
}

```

Скриншот результатов выполнения операции с хеш-таблицей



```

Консоль отладки Microsoft Visual Studio
1 Павлова Лилия Андреевна
5 Николаевна Алена Савельевна
9 Симонов Владимир Тихонович
12 Фетисов Михаил Матвеевич
23 Панкратова Мария Кирилловна
49 Соколова Арина Максимовна
99 Большаков Артем Матвеевич
118 Сахарова Арина Максимовна
121 Поляков Марк Иванович
123 Вектор Дмитрий Иванович
266 Макарова Софья Андреевна
395 Белкина Любовь Максимовна
454 Русакова Виктория Григорьевна
559 Степанов Даниил Николаевич
563 Туманов Егор Алексеевич
619 Фетисов Илья Макарович
793 Трофимова Майя Макарова
855 Смирнова Маргарита Тимофеевна
936 Ильинская Алиса Даниловна

```

Рис.1. Начальное состояние файла

```
Состояние таблицы до удаления записи
key = 559 value = Степанов Даниил Николаевич
key = 1 value = Павлова Лилия Андреевна
key = 121 value = Поляков Марк Иванович
key = 23 value = Панкратова Мария Кирилловна
key = 123 value = Вектор Дмитрий Иванович
key = 5 value = Николаевна Алена Савельевна
key = 266 value = Макарова Софья Андреевна
key = 563 value = Туманов Егор Алексеевич
key = 619 value = Фетисов Илья Макарович
key = 9 value = Симонов Владимир Тихонович
key = 49 value = Соколова Арина Максимовна
key = 12 value = Фетисов Михаил Матвеевич
key = 793 value = Трофимова Майя Макарова
key = 454 value = Русакова Виктория Григорьевна
key = 395 value = Белкина Любовь Максимовна
key = 855 value = Смирнова Маргарита Тимофеевна
key = 936 value = Ильинская Алиса Даниловна
key = 118 value = Сахарова Арина Максимовна
key = 99 value = Большаков Артем Матвеевич
```

Рис.2. Состояние данных файла до удаления строки

Для элементов с ключами 23 и 43, имеющих одинаковый остаток от деления на 20, произошла обработка коллизии, и они успешно встали в таблицу.

```
Измерение времени для вставки первого элемента
key = 559 value = Степанов Даниил Николаевич
key = 1 value = Павлова Лилия Андреевна
key = 121 value = Поляков Марк Иванович
key = 23 value = Панкратова Мария Кирилловна
key = 43 value = Проектория Васильев Иван
key = 5 value = Николаевна Алена Савельевна
key = 266 value = Макарова Софья Андреевна
key = 563 value = Туманов Егор Алексеевич
key = 619 value = Фетисов Илья Макарович
key = 9 value = Симонов Владимир Тихонович
key = 49 value = Соколова Арина Максимовна
key = 12 value = Фетисов Михаил Матвеевич
key = 793 value = Трофимова Майя Макарова
key = 454 value = Русакова Виктория Григорьевна
key = 395 value = Белкина Любовь Максимовна
key = 855 value = Смирнова Маргарита Тимофеевна
key = 936 value = Ильинская Алиса Даниловна
key = 118 value = Сахарова Арина Максимовна
key = 99 value = Большаков Артем Матвеевич
Time: 0.0008ms
```

Рис.3. Вставка первого элемента и оценка времени

```

Измерение времени для вставки последнего элемента
key = 559 value = Степанов Даниил Николаевич
key = 1 value = Павлова Лилия Андреевна
key = 121 value = Поляков Марк Иванович
key = 23 value = Панкратова Мария Кирилловна
key = 43 value = Проектория Васильев Иван
key = 5 value = Николаевна Алена Савельевна
key = 266 value = Макарова Софья Андреевна
key = 563 value = Туманов Егор Алексеевич
key = 619 value = Фетисов Илья Макарович
key = 9 value = Симонов Владимир Тихонович
key = 49 value = Соколова Арина Максимовна
key = 919 value = Короткова Виктория Федоровна
key = 12 value = Фетисов Михаил Матвеевич
key = 793 value = Трофимова Майя Макарова
key = 454 value = Русакова Виктория Григорьевна
key = 395 value = Белкина Любовь Максимовна
key = 855 value = Смирнова Маргарита Тимофеевна
key = 936 value = Ильинская Алиса Даниловна
key = 118 value = Сахарова Арина Максимовна
key = 99 value = Большаков Артем Матвеевич
Time: 64.367ms

find key!find key!Проектория Васильев Иван

```

Рис.3. Вставка последнего элемента и оценка времени

Скриншоты содержания файла и хеш-таблицы во время работы

1	1 Павлова Лилия Андреевна
2	5 Николаевна Алена Савельевна
3	9 Симонов Владимир Тихонович
4	12 Фетисов Михаил Матвеевич
5	23 Панкратова Мария Кирилловна
6	49 Соколова Арина Максимовна
7	99 Большаков Артем Матвеевич
8	118 Сахарова Арина Максимовна
9	121 Поляков Марк Иванович
10	123 Вектор Дмитрий Иванович
11	266 Макарова Софья Андреевна
12	395 Белкина Любовь Максимовна
13	454 Русакова Виктория Григорьевна
14	559 Степанов Даниил Николаевич
15	563 Туманов Егор Алексеевич
16	619 Фетисов Илья Макарович
17	793 Трофимова Майя Макарова
18	855 Смирнова Маргарита Тимофеевна
19	936 Ильинская Алиса Даниловна

Рис.4. Начальное состояние файла

1	559 Степанов Даниил Николаевич
2	1 Павлова Лилия Андреевна
3	121 Поляков Марк Иванович
4	23 Панкратова Мария Кирилловна
5	43 Проектория Васильев Иван
6	5 Николаевна Алена Савельевна
7	266 Макарова Софья Андреевна
8	563 Туманов Егор Алексеевич
9	619 Фетисов Илья Макарович
10	9 Симонов Владимир Тихонович
11	49 Соколова Арина Максимовна
12	919 Короткова Виктория Федоровна
13	12 Фетисов Михаил Матвеевич
14	793 Трофимова Майя Макарова
15	454 Русакова Виктория Григорьевна
16	395 Белкина Любовь Максимовна
17	855 Смирнова Маргарита Тимофеевна
18	936 Ильинская Алиса Даниловна
19	118 Сахарова Арина Максимовна
20	99 Большаков Артем Матвеевич

Рис.5. Конечное состояние файла

Выводы

В результате проделанной работы, я получила навыки по разработке хеш-таблиц и их применении. В процессе было разработано приложение, которое использует хеш-таблицу для организации прямого доступа к записям файла.

Список информационных источников

1. Лекционный материал по структурам и алгоритмам обработки данных Сартакова М. В. (дата обращения 15.09.2021)
2. Дополнительный материал к практическим работам по структурам и алгоритмам обработки данных Сорокина А. В. (дата обращения 15.09.2021)