Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Ордена Трудового Красного Знамени

федеральное государственное бюджетное образовательное учреждение высшего образования

МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И ИНФОРМАТИКИ

Кафедра «Математической кибернетики и информационных технологий»

Информационные технологии и программирование

Лабораторная работа №3

Выполнил: студент группы

БВТ2306

Кесслер Алексей Сергеевич

Москва, 2024 г.

Цель работы: Изучение структуры данных Hash Table, хэш-функций, основ работы с обобщениями

Задача работы: Создание собственной структуры Hash Table с решением коллизий с помощью метода цепочек.

Выполнение

Задание 1: Создать класс HashTable, который будет реализовывать хэш-

таблицу с помощью метода цепочек.

Объявляем класс HashTable с дженериками, задаем ему capacity и создаем динамический массив со связанными списками-цепочками

```
public class HashTable <T, K> {

    4 usages
    private final int hashTableSize = 100;
    4 usages
    private boolean isEmpty;
    5 usages
    private int amountOfElements;
    10 usages
    public ArrayList<LinkedList<HashTableEntry<T, K> > > tableIndex;
```

Рисунок 1 - Класс HashTable

```
public HashTable() {

    isEmpty = true;
    amountOfElements = 0;
    tableIndex = new ArrayList <LinkedList<HashTableEntry<T, K> > >();
    for (int i = 0; i < hashTableSize; i++) {
        tableIndex.add(null);
    }
}
```

Рисунок 2 - Конструктор

Теперь реализуем метод put, который вставляет значения в нашу хэш таблицу. Принцип его работы таков: сначала он хэширует объект(хэш-функцию необходимо перегрузить), берет значению по модулю, чтобы можно было получить индекс в динамическом массиве. Далее мы туда просто вставляем значение, если его не было, иначе устанавливаем ему значение в его месте в LinkedList.

```java
public void put (T key, K value) {
    int index = key.hashCode() % hashTableSize;

    isEmpty = false;
    if (tableIndex.get(index) == null) {
        tableIndex.set(index, new LinkedList<HashTableEntry<T, K>>());
    }
    for (HashTableEntry<T, K> entry : tableIndex.get(index)) {
        if (entry.equalsKeys(key)) {
            entry.setValue(value);
            return;
        }
    }
    amountOfElements += 1;
    tableIndex.get(index).add(new HashTableEntry<T, K>(key, value));

}
```

Рисунок 3 - Метод put

Метод get делает ровно то же самое, только он получает значение, а не устанавливает

```java
public K get (T key) {
    int index = key.hashCode() % hashTableSize;

    if (tableIndex.get(index) == null) {
        return null;
    }
    for (HashTableEntry<T, K> entry : tableIndex.get(index)) {
        if (entry.equalsKeys(key)) {
            return entry.getValue();
        }
    }
    return null;
}
```

Рисунок 4 - Метод get

И метод remove тоже делает то же самое, но только убирает значение из массива

```java
public void remove (T key) {
    int index = key.hashCode() % hashTableSize;

    for (HashTableEntry<T, K> entry : tableIndex.get(index)) {
        if (entry.equalsKeys(key)) {
            amountOfElements -= 1;
            tableIndex.get(index).remove(entry);
        }
    }
    if (amountOfElements == 0) {
        isEmpty = true;
    }
}
```

Рисунок 5 - Метод remove

И так же напишем дополнительные методы size и isEmpty

```java
no usages  DaNKAadnka
public int size() { return this.amountOfElements; }

no usages  DaNKAadnka
public boolean isEmpty() { return this.isEmpty; }
```

Рисунок 6 - Методы size и isEmpty

Задание 2. Мой вариант - 5: Реализация хэш-таблицы для учета продуктов на складе.

Ключом будет штрихкод товара, а значением - объект класса Product,

содержащий информацию о названии, цене и доступном количестве.

Необходимо реализовать операции вставки, поиска и удаления продукта

по штрихкоду.

Для начала реализуем структуру штрихкода:

```java
public class BarCode {

    8 usages
    public String code;

    4 usages  DaNKAadnka
    public BarCode(String s) { code = s; }
```

Рисунок 7 - Структура штрихкода

Для нее переопределим хэш функцию и метод сравнения значений:

```java
@Override
public int hashCode() {
    int b = 31;
    int mod = 115249;
    long hash = 0;
    for (int i = 0; i < code.length(); i++) {
        hash += (binPower(b, i, mod) * (code.charAt(i) - '0')) % mod;
    }

    return (int)hash;
}

no usages  DaNKAadnka
public boolean equals(BarCode a) {
    boolean eqs = true;
    if (a.code.length() != code.length()) {
        eqs = false;
    }
    for (int i = 0; i < code.length(); i++) {
        if (a.code.charAt(i) != code.charAt(i)) {
            eqs = false;
        }
    }
    return eqs;
}
```

Рисунок 8 - Хэш функция и метод equals

И реализуем простой класс Product

```
public String productName;
1 usage
public int productPrice;
1 usage
public int productAvailableAmount;

4 usages  DaNKAadnka
public Product(String name, int price, int amount) {
    productName = name;
    productPrice = price;
    productAvailableAmount = amount;
}
```

Рисунок 9 - Класс Product

Теперь мы вполне можем использовать хэш таблицу с этими классами:

```java
HashTable<BarCode, Product> ht = new HashTable<~>();

BarCode b1 = new BarCode( s: "1238462048274");
BarCode b2 = new BarCode( s: "7391837462917");
BarCode b3 = new BarCode( s: "8937462840183");

ht.put(b1,
        new Product( name: "Papaya",  price: 120,  amount: 1000));

ht.put(b2,
        new Product( name: "Marakuya",  price: 150,  amount: 500));

ht.put(b3,
        new Product( name: "Avacado",  price: 80,  amount: 3000));

ht.put(b1,
        new Product( name: "NoPapaya",  price: 120,  amount: 1000));

Product guessWho1 = ht.get(b1);
Product guessWho2 = ht.get(new BarCode( s: "0192847593872"));


System.out.println(guessWho1.productName);
if (guessWho2 == null) System.out.println("null");

ht.remove(b1);

if (ht.get(b1) == null) {
    System.out.print("We are now without Papaya");
}
```

Рисунок 10 - Использование хэш таблицы для учета продуктов.