Algorithms With Java

Week 2 Day 2: HashSet

Академия Ковалевского

Содержание

1. Теория

Сегодня мы сведем воедино то, что делали в прошлых днях и реализуем несколько практических задач. В реальной жизни разные реализации одного и того же типа структуры данных адаптированы для разных операций.

Например LinkedList имеет всегда сложность O(1) на добавление нового элемента в начало и конец списка, что делает его идеальным для реализации очереди. В то время как ArrayList имеет худшую сложность добавления в конец O(N) и постоянную сложность O(N) при добавлении в начало, зато, в отличии от LinkedList, он имеет всегда сложность O(1) на чтение элемента из любого индекса.

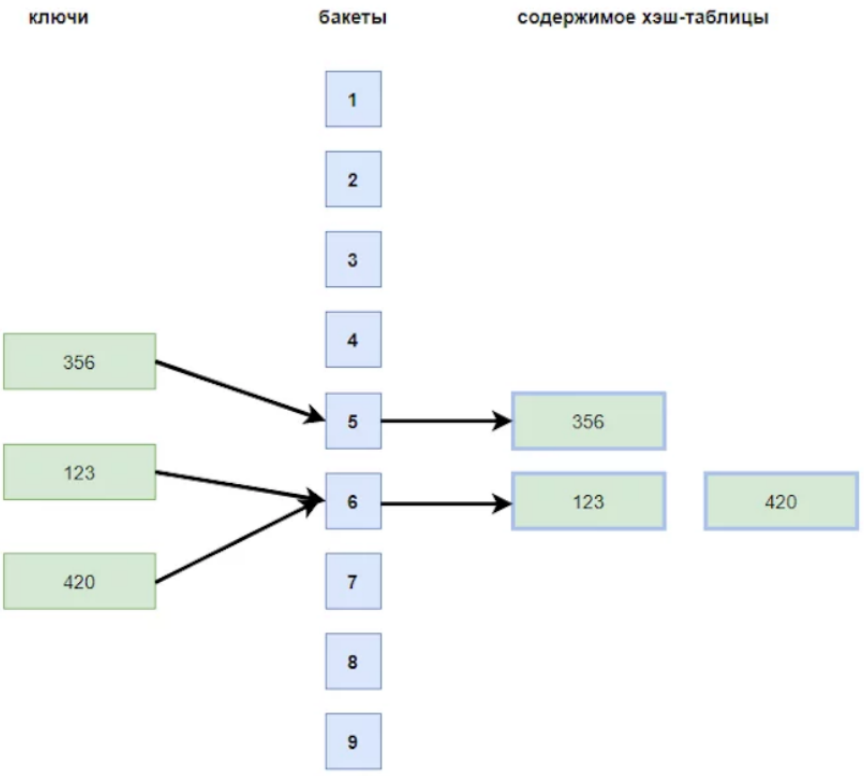
HashSet

Сегодня мы реализуем свой собственный Set, который хранит числа и использует для этого деревья созданные ранее. Благодаря этому наш Set будет иметь преимущества в сравнении с HashSet из стандартной библиотеки Java, в частности он будет иметь константную сложность подсчета суммы O(k), где k — это число бакетов в HashSet.

Set — это неупорядоченное множество уникальных элементов. Например, мешочек с бочонками для игры в лото: каждый номер от 1 до 90 встречается в нем ровно один раз, и заранее неизвестно, в каком порядке бочонки вынут при игре.

HashSet — это конкретная реализация *Set*. Класс *HashSet* использует для хранения данных хеш-таблицу. Хеш-таблица — структура данных, в которой все элементы помещаются в бакеты (buckets), соответствующие результату вычисления хеш-функции. Это значит, что при манипуляциях с элементами используется хеш-функция — *hashCode()*.

Например, администратор в гостинице может класть ключ в коробку с номером от 1 до 9, вычисляя его по такому алгоритму: складывать все цифры номера, пока не получится одноразрядное число. Здесь алгоритм вычисления — хеш-функция, а результат вычисления — хеш-код. Тогда ключ от номера 356 попадёт в коробку 5 (3 + 5 + 6 = 14; 1 + 4 = 5), а ключ от номера 123 — в коробку с номером 6. Хеш таблица примера:



В каждом бакете нужно будет хранить дерево. Таким образом у нас будет K деревьев вместо одного большого дерева, и как результат в среднем размер каждого дерева будет N/K, где N — это число элементов во всем Set, а K — это число бакетов.

Scraping

Scraping — это извлечение открытой публичной информации с многих сайтов за короткое время. Статья на вики о веб-скрейпинге. Примеры применения скрейпинга:

- Найти информацию об определенной категории продуктов с ценами на разных сайтах.
- Если нет интернета — можно скачать сайт в структурированном виде.
- Можно автоматически собирать данные о вакансиях.
- Если есть желание сэкономить — скрейпинг тоже может помочь найти товар со скидкой.

ВАЖНО: Для решения этой задачи нужно использовать Java HTML Parser — Jsoup. Подключите его в maven проект!

2. Практическая работа

IntSet

Это класс реализует HashSet, который хранит целочисленные значения. Он имплементирует алгоритм хэшей для оптимизирования скорости нахождения элементов в сете. В этом классе нужно реализовать:

1. Конструктор IntSet(int bucketsSize) — создаёт количество бакетов, принимаемое в качестве аргумента.
2. Конструктор по умолчанию IntSet() — создает три бакета.
3. Метод getBucketsCount() — возвращает количество бакетов в этом сете которое было создано в конструкторе.
4. Метод getBucketId() — на вход принимает объект и возвращает идентификатор бакета, в котором находится этот объект. Важно: id бакета это остаток от деления числа на количество бакетов. Например, если у нас 3 бакета то бакет для числа 0 - 0, для числа 3 - 0, для числа 4 - 1, для числа 1 - 1, и т.д.
5. Метод contains() — говорит, содержит ли сет заданное значение.
6. Метод add() — добавляет в сет заданное значение
7. Метод count() — считает количество элементов в сете.
8. Метод sum() — считает сумму значений всех элементов.

```
package academy.kovalevskyi.algorithms.week2.day2;

import academy.kovalevskyi.algorithms.week2.day0.IntTreeHelper;
import academy.kovalevskyi.algorithms.week2.day0.IntTreeNode;

public class IntSet {

    public IntSet(int bucketsSize) {
      // TODO
    }

    public IntSet() {
      // TODO
    }

    public int getBucketsCount() {
```

```
    // TODO
  }

  public int getBucketId(int value) {
    // TODO
  }

  public boolean contains(int value) {
    // TODO
  }

  public void add(int value) {
    // TODO
  }

  public int count() {
    // TODO
  }

  public int sum() {
    // TODO
  }
}
```

WikiScrapper

Из библиотеки jsoup разрешено использовать только:

- **Jsoup.connect**(url).**get**();
- doc.**select**("body").**first**()
- doc.**children**()
- element.**text**()

Это важно, так как заставит вас обходить дерево элементов страницы. Данный класс на вход будет получать страницу из WikiPedia (например: ссылка) и должен:

- Найти на ней тег, в котором указана дата последнего обновления этой страницы
- Получить эту дату и преобразовать ее в Date

В этом классе нужно реализовать:

1. Конструктор, который должен иметь модификатор доступа private, чтобы объект нельзя было создавать напрямую.
2. Метод generateScrapper() — создает объект WikiScrapper.
3. Метод lastEditedOn() — вытягивает со страницы Википедии дату последнего обновления статьи.

```
package academy.kovalevskyi.algorithms.week2.day2;

import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;


public class WikiScrapper {

  public String getUrl() {
    // TODO
  }
```

```
  public static WikiScrapper generateScrapper(String page) throws IOException
{
    // TODO
  }

  public Date lastEditedOnDate() throws ParseException {
    // TODO
  }
 }
```