Prof. Dr. Ralf Hinze
Markus Heinrich, M.Sc.

TU Kaiserslautern
Fachbereich Informatik
AG Programmiersprachen

Miniprojekt 2: Programmierpraktikum 2021

Ausgabe: 25. Mai 2021
Abgabe: 8. Juni 2021, 15 Uhr

GitLab Team Repositories

Wir verwenden auch für das zweite Miniprojekt wieder die GitLab Repositories. Führen Sie den Befehl `git pull` aus, damit die Vorlagen heruntergeladen werden. Sie sollten nun einen neuen Ordner `MP2` sehen, den Sie wie gewohnt in IntelliJ IDEA öffnen können. Es handelt sich wieder um ein Gradle Projekt, sodass Abhängigkeiten automatisch heruntergeladen werden.

Ihre Lösungen müssen Sie in das lokale Repository committen (`git commit`) und dann auf den GitLab Server übertragen (`git push`). Sprechen Sie sich zur Bearbeitung der Aufgaben mit Ihren Abgabepartner*innen ab, damit Sie unnötige Konflikte vermeiden.

KWIC-Index

Mit Hilfe eines KWIC-Indexes (**k**ey**w**ord **i**n **c**ontext) ist es bei einer Literaturrecherche möglich, gezielt zu einem Stichwort Artikel oder Bücher herauszusuchen. Wenn die folgenden Titel gegeben sind,

```
Introduction to Functional Programming
The implementation of functional programming languages
```

dann enthält der KWIC-Index folgende Einträge.

```
Functional Programming. Introduction to
functional programming languages. The implementation of
implementation of functional programming languages. The
Introduction to Functional Programming.
languages. The implementation of functional programming
Programming. Introduction to Functional
programming languages. The implementation of functional
```

Die Titel werden wortweise rotiert. Die auf diese Weise generierten Strings werden alphabetisch sortiert, wobei nicht zwischen Kleinbuchstaben und Großbuchstaben unterschieden wird. Aus der Liste werden darüber hinaus alle Einträge gestrichen, die mit einem uninformativen Wort, z. B. einem Artikel oder einer Präposition, beginnen. Ein Punkt markiert in der Ausgabe das Ende eines Titels, so dass der ursprüngliche Titel wieder rekonstruiert werden kann.

Wie Sie bei der Implementierung eines Tries in Miniprojekt 1 gesehen haben, eignet sich diese Datenstruktur dazu, eine effiziente Präfixsuche zu realisieren. Diese Eigenschaft möchten wir uns zunutze machen, um unseren KWIC-Index zu durchsuchen.

In der Praxis macht man bei Programmieraufgaben häufig von bereits existierenden Programmpaketen gebrauch, die das eigene Problem oder eventuell sogar ein allgemeineres Problem lösen. In diesem Fall werden wir das Paket *concurrent-trees*[1] verwenden. Solche Abhängigkeiten des eigenen Projekts lassen sich mithilfe von Gradle verwalten. Die Abhängigkeiten werden in der Datei `build.gradle` hinterlegt und Gradle lädt das entsprechende Paket selbstständig von einem zentralen Repository herunter (achten Sie auf das Ladefenster, wenn Sie das Projekt zum ersten Mal öffnen).

[1] https://github.com/npgall/concurrent-trees

a) Implementieren Sie zunächst die Klasse `public class TrieMap<V> implements Trie<V>`. Wie bereits angedeutet, sollen Sie hier keine eigene Trie Implementierung verwenden. Verwalten Sie stattdessen einen `ConcurrentRadixTree` (Schnittstelle `RadixTree`) aus dem Paket `com.googlecode.concurrenttrees.radix` in Ihrer `TrieMap` Implementierung und rufen Sie in den Methoden der `TrieMap` entsprechende Methoden von `RadixTree` auf (schauen Sie dazu in die Dokumentation). `TrieMap` kann also als sogenannte *wrapper*-Klasse verstanden werden, welche die Funktionalität einer anderen Klasse kapselt.

   Die Schnittstelle `Trie` ist im Vergleich zu Miniprojekt 1 leicht abgewandelt:

```
public interface Trie<V> {
    V get(String key);
    void put(String key, V value);
    void remove(String key);
    int size();
    Iterable<KeyValuePair<V>> searchKeyPrefix(String prefix);
}
```

   Wie Sie sehen, verwenden wir zur Repräsentation von Schlüssel-Wert-Paaren die Klasse `KeyValuePair`. Informationen dazu finden Sie ebenfalls in der Dokumentation.

   Sichern Sie als Wert zu den rotierten Titeln das entsprechende `Book` Objekt (verwenden Sie also eine `TrieMap<Book>`).

   Hinweis: Der Konstruktor von `ConcurrentRadixTree` verlangt eine Node Factory. Sie können einfach `new DefaultCharArrayNodeFactory()` übergeben.

   Hinweis: Zu Testzwecken kann es hilfreich sein, sich den Trie als String zurückgeben zu lassen. Überschreiben Sie dazu die Methode `toString()` (die jede Java-Klasse von der `Object` Klasse erbt) und rufen Sie dort `PrettyPrint.prettyPrint` auf (s. Dokumentation).

   Information: Wie der Name `ConcurrentRadixTree` vermuten lässt, unterstützt die Implementierung lockfreie Lesezugriffe und erlaubt nebenläufige Schreibzugriffe. Davon machen wir im Rahmen des Miniprojekts jedoch keinen Gebrauch.

b) Implementieren Sie die Klasse `public class KwicImpl implements Kwic`. Bücher repräsentieren wir durch die Klasse `Book`.

```
public interface Kwic {
    void add(Book book);
    List<Book> search(String term);
}
```

   Denken Sie daran, nur solche Rotationen des Titels in den Index aufzunehmen, die nicht mit einem uninformativen Wort beginnen. Eine entsprechende Sammlung finden Sie in der `Util` Klasse im Attribut `exceptions`.

   Die Suche soll unabhängig von Groß- und Kleinschreibung funktionieren, wandeln Sie die rotierten Buchtitel daher beim Einfügen in den Index und später bei der Suche zum Beispiel in Kleinbuchstaben um (`toLowerCase`).

   Hinweis: Um die eingangs erwähnte Sortierung der rotierten Titel müssen Sie sich nicht kümmern, das erledigt die Klasse `ConcurrentRadixTree`.

c) In der `Util` Klasse wird im Array `public static final Book[] books` eine kleine „Bibliothek“ mitgeliefert. Fügen Sie diese zunächst in der `main` Methode der `Main` Klasse in einen KWIC-Index ein und fordern Sie die Benutzerin/den Benutzer des Programms auf, einen Suchbegriff einzugeben. Geben Sie Treffer im Format `Autor. Titel. Jahr.` auf der Konsole aus.