# Beleg 2

Ziel dieses Belegs ist es die Anwendung von Higher Order Functions in Zusammenhang mit unterschiedlichen Datenstrukturen zu üben. Ausgangspunkt der Übung sind Textdateien, für die verschiedenste Statistiken erstellt werden sollen. Weiter sollen die Dateien effizient durchsuchbar gemacht werden. Das Grundgerüst für alle zu schreibenden Funktionen sowie verschiedene Tests finden Sie in dem Beleg-Projekt, dass unter plus.htw-berlin.de heruntergeladen werden kann.

In dem Projekt befinden sich im Testressourcen-Ordner verschiedene Beispieldateien, die für die Analyse verwendet werden können/sollen: MobyDick.txt (Buch Moby Dick von Herman Melville), MobyDickC1.txt (Kapitel 1 von Moby Dick), MobyDickShort.txt (Teil aus dem ersten Kapitel), Robinson.txt (Buch “Robinson Crusoe” von Daniel Defoe) sowie AFINN-111.txt (Beispieldatei für die Sentimentanalyse).

Verwenden Sie für den Beleg keine Variablen – nur Elemente die sie bisher aus der Funktionalen Programmierung kennen gelernt haben.

Aufgabe 1: Vervollständigen Sie die Funktionen *getWords*, *getAllWords* und *countWords*, die das Zählen von Wörtern innerhalb eines Texts ermöglichen sollen. Der Text befindet sich dabei im folgenden Format: Jede Zeile besteht aus einem Tupel (Int, String) das eine Zeilennummer enthält sowie die Textzeile als String. Beim Zählen der Wörter soll keine Unterscheidung zwischen Groß- und Kleinschreibung gemacht werden, d.h. Baum wäre bspw. das selbe Wort wie bauM. Gehen Sie dabei folgendermaßen vor:
Implementieren Sie im ersten Schritt die Funktion *getWords*, die aus einem String alle Wörter extrahiert. Ersetzen Sie zu diesem Zweck zuerst alle Element, die keine Buchstaben sind, durch das Leerzeichen und splitten Sie dann den String auf Basis der Leerzeichen (Hinweis: Schauen Sie sich die Funktionen *split* und *replaceAll* dafür an)
Implementieren Sie dannach die Funktion getAllWords, die aus dem Gesamttext (Liste von Tupeln (Zeilennr, Text) alle Wörter extrahiert.
Sind die ersten beiden Funktionen implementiert, so kann eine Liste aller im Text vorkommenden Wörter generiert werden, welche wiederum Input der Funktion *countWords* ist. Sie zählt alle Vorkommen eines Wortes und gibt eine Liste bestehend aus den Wörtern und deren Anzahl zurück. Verwenden Sie dafür die Datenstruktur Map.

Aufgabe 2: In dieser Aufgabe sollen große Texte durchsuchbar gemacht werden, d.h. es soll nach Zeilen gesucht werden können, in denen beliebig gewählte Schlüsselwörter vorkommen. Dazu soll im ersten Schritt ein sogenannten Inverser Index erstellt werden. Im Invesen Index soll gespeichert werden, in welchen Zeilen innerhalb eines Dokuments ein Wort vorkommt. Implementieren Sie hier zunächst die Funktion getAllWordsWithIndex. Diese Funktion bekommt den Text im Format List((Zeilennr, Zeilentext)), extrahiert alle Wörter und speichert diese als Tupel in der Form (Zeilennr, Wort). Jetzt müssen in der Funktion createInverseIndex sämtliche Vorkommen eines Wortes (Zeilennummern) innerhalb einer Map zusammengefasst werden. Ergebnis der Funktion ist somit eine Map, die ein Wort auf eine Liste von Zeilennummern (Int) abbildet.

Auf dieser Basis können jetzt die Funktionen andConjunction und orConjunction implementiert werden. Ziel der Funktionen ist es, für eine Liste von Wörtern die Zeilen herauszusuchen, die diese enthalten. Dabei sollen bei orConjunction die Wörter “verodert” werden und diese Zeilen zurückgegeben werden, in die mindestens eins der Wörter vorkommt. Bei andConjunction sollten die Wörter mit dem logischen und verknüpft werden und nur Zeilen zurückgegeben werden, in denen alle Wörter vorkommen.

Aufgabe 3: In dieser Aufgabe soll eine Sentiment-Analyse durchgeführt werden. In einer Sentiment-Analyse werden gefühlsausdrückenden Wörtern ein Wert von +5 (positiv) bis -5 (negativ) zugeordnet. Diese Zuordnung befindet sich in der Datei AFINN-111.txt. In der vorgegebenen Funktion *getSentiments* wird diese Datei ausgelesen und im Format einer Map [String, Int] bereitgestellt. Auf Basis dieser Sentiment-Zuordnung soll eine Analyse durchgeführt werden. Implementieren Sie hierzu als erstes die Funktionen *getDocumentGroupedByCounts.* Diese bekommt als Parameter einen Dateinamen sowie die Anzahl von Wörtern, die innerhalb eines Abschnitts zusammengefasst werden sollen. Ergebnis der Funktion ist dann ein Tupel, bestehend aus der Abschnittsnummern und einer Liste von Wörtern, die dort vorkommen. Die Abschnittsnummern sollten bei 1 Anfangen.

Implementieren sie jetzt eine Funktion *GetDocumentSplitByPredicate*. Diese ist ähnlich zu *getDocumentGroupedByCounts* und bekommt jedoch als Parameter einen Dateinamen sowie ein Predikat (Funktion mit booleschen Rückgabewert), das benutzt wird um die Abschnitte zu definieren. Jedes Mal wenn die Funktion für eine Zeile „True“ zurückgibt, sollte ein neuer Abschnitt begonnen werden. (anstatt nachdem n-Wörter gesammelt worden sind, wie bei der Funktion davor) Der Text, der vor dem ersten Abschnitt ist sollte verworfen werden. Die Funktion hat den gleichen Rückgabewert wie *getDocumentGroupedByCounts*.

Letzte zu implementierende Funktion ist analyseSentiments. Diese bekommt eine Liste von Abschnitten (Abschnittnr, Liste von Wörtern) und errechnet den jeweiligen Sentimentwert. Der Sentimentwert wird als der Durchschnitt der Sentimentwerte aller bekannten Wörtern in dem Abschnitt berechnet. Ergebnis ist ein Tripel, bestehend aus der Abschnittsnummer, dem Sentimentwert und der relativen Anzahl von Wörtern, die für die Sentimentanalyse verwendet werden konnten.

In der Klasse App befindet sich eine Mainklasse, die für das Buch „Robinson Crusoe“ die Ergebnisse der Sentimentanalyse graphisch darstellt. Testen. Sie diese und überprüfen Sie die Plausibilität der Ergebnisse.

Die Belegarbeit kann in Gruppen von 1-2 Personen bearbeitet werden und ist bis zum 13.12.2019 in der Übung abzugeben. Die Funktionen sollten mindestens die mitgelieferten Tests bestehen. Es ist ratsam, weitere Tests hinzuzufügen. Die Abnahme der Belegaufgabe erfolgt in der Übung.