

IR-Grundlagen - Worksheet 2

Dieses Worksheet behandelt den Bi-Word Index und den Positional Index. Das Ziel des Worksheets ist es Mehr-Wort Queries sowie Proximity Suchen (z.B: Term a im Umkreis von 3 Termen von Term b) durchzuführen. Sie können für dieses Sheet entweder das bereitgestellte Eclipse-Project verwenden oder ihre Lösung des letzten Worksheets erweitern.

Grundsätzliche Struktur:

Die notwendige Implementierung befindet sich ausschließlich in den `Index`-Klassen. Sie können den `MainController` nutzen um Objekte der Indexe zu erzeugen und eigene Suchanfragen an die Indexe zu stellen. Der `MainController` ist nicht Teil der Bewertung

Zuerst beschäftigen wir uns mit dem Bi-Word Index. Dieser ist dem Inverted-Index des letzten Sheets sehr ähnlich nutzt aber Kombinationen von benachbarten Wörtern als Terme die indexiert werden:

1. Die neue Klasse `BiWordIndex` ähnelt vom Konzept her der `InvertedIndex` Klasse vom letzten Arbeitsblatt. Der Unterschied besteht darin dass der Bi-Word-Index jeweils zwei benachbarte Terme als einen Index-Term verwenden soll.
 - a. Nutzen sie die zur Verfügung stehenden `public` Methoden der `Document`-Klasse um den Index zu befüllen. Erledigen Sie diese Arbeit im Konstruktor der Bi-Word Index-Klasse. Schauen sie sich ihren Code der `InvertedIndex` Klasse an und überlegen sie sich welche Teile davon sie verwenden und welche sie modifizieren müssen. Legen sie bei Bedarf weitere Methoden und Variablen an.
(Tipp: sie können ihre `Inverted-Index` Klasse als super-Klasse zum Bi-Word Index verwenden)



- b. Die Methode

```
ArrayList<Integer> searchForSingleWord(String[] word)
```

kann unverändert vom letzten Worksheet übernommen werden. Überlegen sie sich warum.

Während der Bi-Word Index dem Inverted Index noch relativ ähnlich war befassen wir uns jetzt mit einem etwas komplizierteren Ansatz, dem Positional Index.

2. Im Skript befindet sich folgende Definition des Positional Index:

```
<term, number of docs containing term;  
doc1: position1, position2 ... ;  
doc2: position1, position2 ... ;  
etc.>
```

Wobei 'term' ein Wort in der collection darstellt, 'number of docs' die Anzahl der Dokumente die 'term' beinhalten und 'doc1: position1, position2, ...' eine Liste von Positionen an denen 'term' in 'Document 1' vorkommt.


Versuchen sie in der neuen Klasse `PositionalIndex` diese Struktur zu repräsentieren. Überlegen sie sich eine Datenstruktur die Alle diese Informationen für ALLE Terme speichern kann. Erstellen sie wenn notwendig selbstständig neue Klassen und verwenden sie geeignete Datenstrukturen. Wie in den vorherigen Aufgaben bekommt der Konstruktor der Klasse eine `ArrayList (collection)` mit `BooleanDocument`-Objekten.

(Nützlich, aber nicht zwingend notwendig: `TreeMap` /


<https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html>)

3. Interaktion mit dem Positional Index

Um den Positional Index zu nutzen sowie zu testen müssen sie nun mehrere Methoden die vordefiniert sind implementieren.

 a. `public ArrayList<Integer> searchForSingleWord
(String word)`

Diese Methode gibt eine Liste von DocId's der Dokumente, die diesen Term beinhalten, zurück.

 b. `public ArrayList<Integer> searchForSingleWordInDocument
(String word, int docNumber)`

Diese Methode gibt eine Liste von Positionen, an denen der Term 'word' im Dokument mit der ID 'docNumber' vorkommt, zurück.



c. `public ArrayList<Integer> searchForPhrase
(String[] phrase)`


Diese Methode gibt eine Liste von Dokumenten-ID's der Dokumente zurück in denen die einzelnen Wörter der Query (`phrase`) in der genauen Reihenfolge in denen sie eingegeben wurden vorkommen.

Implementieren sie diese Methode zuerst für 2-Wort Phrasen. In Aufgabe 5 erweitern wir diese Methode für beliebig viele Wörter.


Sie können sich am folgenden Pseudocode orientieren. Achten sie aber darauf dass der Pseudocode für die Optionale Aufgabe 5 und 6 (Proximity-Search) vorgesehen ist. Entscheiden sie welche Teile des Pseudocodes sie benötigen.

```
POSITIONALINTERSECT( $p_1, p_2, k$ )
1   $answer \leftarrow \langle \rangle$ 
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3  do if  $\text{docID}(p_1) = \text{docID}(p_2)$ 
4      then  $l \leftarrow \langle \rangle$ 
5           $pp_1 \leftarrow \text{positions}(p_1)$ 
6           $pp_2 \leftarrow \text{positions}(p_2)$ 
7          while  $pp_1 \neq \text{NIL}$ 
8          do while  $pp_2 \neq \text{NIL}$ 
9              do if  $|\text{pos}(pp_1) - \text{pos}(pp_2)| \leq k$ 
10                 then  $\text{ADD}(l, \text{pos}(pp_2))$ 
11                 else if  $\text{pos}(pp_2) > \text{pos}(pp_1)$ 
12                     then break
13                      $pp_2 \leftarrow \text{next}(pp_2)$ 
14                 while  $l \neq \langle \rangle$  and  $|l[0] - \text{pos}(pp_1)| > k$ 
15                     do  $\text{DELETE}(l[0])$ 
16                     for each  $ps \in l$ 
17                         do  $\text{ADD}(answer, \langle \text{docID}(p_1), \text{pos}(pp_1), ps \rangle)$ 
18                      $pp_1 \leftarrow \text{next}(pp_1)$ 
19                  $p_1 \leftarrow \text{next}(p_1)$ 
20                  $p_2 \leftarrow \text{next}(p_2)$ 
21             else if  $\text{docID}(p_1) < \text{docID}(p_2)$ 
22                 then  $p_1 \leftarrow \text{next}(p_1)$ 
23                 else  $p_2 \leftarrow \text{next}(p_2)$ 
24 return  $answer$ 
```

Optionale Aufgaben:

-  4. Erweitern sie die `searchForPhrases` Methode der Teilaufgabe 4.c damit nach beliebig langen Queries gesucht werden kann.

Sie können sich hierfür am Pseudocode von Aufgabe 4.c orientieren

-  5. Implementieren Sie die überladene Version der `searchForPhrase` Methode so dass eine Suche für einen Term im Umkreis von x Termen eines anderen Terms möglich ist. Zwei benachbarte Terme haben dabei den Abstand 1.

```
public ArrayList<Integer> searchForPhrase  
    (String[] phrase, int k)
```

Sie können sich hierfür am Pseudocode von Aufgabe 4.c orientieren