**Programmierkurs Python I – WS 10/11**

# Programmierprojekt

---

Ziel des Projekts ist es, eine kleine Suchmaschine zu implementieren. Es sollen Suchanfragen formuliert werden können, die mit den üblichen Boolschen Operatoren verknüpft sein können. Das Ergebnis soll eine Liste von Dokumenten sein, die die entsprechende Suchanfrage erfüllen.

Das Projekt gliedert sich in zwei Teilprojekte:

1. Erstellen eines Index über der Dokumentsammlung. Der Index soll für jedes Wort eine Liste von Dokumenten (genauer: deren Namen/ID) enthalten, in denen das entsprechende Wort vorkommt. Der Index soll als `shelve`-Objekt gespeichert werden (siehe Vorlesung). In der Dokumentation soll stehen, welche Datei den Suchindex enthält; man soll ihn sich getrennt von der übrigen Programmfunktionalität betrachten können.

2. Implementieren einer Funktion, mit der man Suchanfragen über dem Index formulieren kann. Das Programm soll so aufgebaut sein, dass man Suche und Index-Erstellung getrennt voneinander benutzen kann!

Neben dem Programm-Code soll auch eine kurze **Dokumentation** angefertigt werden, die die Funktionsweise kurz illustriert (Umfang: etwa eine Seite).

## 1 Erstellen des Suchindex

Der Suchindex soll aus dem New-York-Times-Teil des *Gigaword*-Korpus erstellt werden. Die Dokumente befinden sich nach Jahrgang und Monat aufgeteilt unter `/proj/corpora/en_gigaword/nyt` auf den CoLi-Servern. Zur Erstellung des Index müsst ihr die Dokumente einlesen, deren Wörter auszählen und die relevanten Wörter für jedes Dokument herausfinden:

1. Die Dokumente liegen in einem XML-artigen Format vor, in gepackten Dateien. Die Dateien enthalten jeweils mehrere Dokumente (Tag: DOC). Jedes Dokument hat einen Namen (Attribut: id) und einen Typ (Attribut: type), der angibt, um was für eine Art Dokument es sich handelt. Es sollen jeweils nur Dokumente mit Typ „story“ betrachtet werden.

2. Zur Bestimmung der relevanten Wörter benutzen wir tf-idf[1]. Dafür müsst ihr zunächst für jedes Wort in der Dokumentsammlung bestimmen, in wie vielen

[1] Eine Beschreibung findet sich unter `http://en.wikipedia.org/wiki/Tf-idf`.

Dokumenten es (egal wie oft) vorkommt. Danach müsst ihr für jedes einzelne Dokument die genauen Worthäufigkeiten auszählen und daraus den tf-idf-Wert berechnen.

3. In dem Index, der am Ende gespeichert werden soll, soll jedes Wort auf alle Dokumente abgebildet werden, für die es relevant ist. (Unter *war* sollen alle Dokumente auffindbar sein, für die *war* eines der relevanten Worte ist.) Um den Index möglichst klein zu halten, sollen der Einfachheit halber nur Wort-Dokument-Paare im Index aufgeführt werden, bei denen das Wort eines der 10 relevantesten Wörter des Dokuments ist. (Unter *war* stehen die Dokumente, für die *war* unter den 10 Worten mit dem höchsten tf-idf-Wert war.) Es reicht hier völlig aus, die Dokument-IDs zu Speichern (nicht den vollständigen Text).

4. Satzzeichen sollen ignoriert werden.

## Hinweise:

Die Dokumentsammlung ist sehr groß, daher empfiehlt es sich, das Programm nur mit einer einzigen Datei zu testen. Die Berechnung des Index für alle Dokumente kann sehr lange dauern (abhängig von der konkreten Implementierung durchaus mehrere Tage). Es bietet sich daher an, die Berechnung auf einem der Cluster-Rechner auszuführen, und nur auf einer Teilmenge der Dokumente. Bei Verwendung des Unix-Werkzeugs „screen“ kann man sich abmelden, ohne dass die Berechnung unterbrochen wird. (Alternativ: nohup).

Es bietet sich an, einen Cluster-Rechner (`cluster-1` bis `cluster-16`) zu verwenden. Um sich auf diesen Rechnern anmelden zu können, muss man sich innerhalb des Coli-Netzwerks befinden. Wenn Du dich von außerhalb anmelden möchtest, musst Du dich über den Rechner `login` anmelden. Anmelden kannst Du dich per `ssh`. Auf MaxOS X und Linux-Rechnern ist `ssh` normalerweise vorinstalliert. Für Windows-Rechner gibt es verschiedenen (kosten-) freie SSH-Implementierungen.

Unter Umständen werden für den Index sehr große Dateien erzeugt. Bitte speichere diese Datei(en) nicht im Home-Verzeichnis. Unter `/local` kannst Du dir ein Verzeichnis auf dem Cluster direkt anlegen, und dort die Dateien speichern. (Nach beenden des Projektes solltest Du diese Dateien wieder löschen.)

**Evaluation dieses Zwischenschrittes:** In unserer Referenz-Implentierung, getestet mit dem Dokument `nyt199411.gz`, gibt es 25318 Einträge im Suchindex, also 25318 Suchworte, für die die relevanten Dokumente eingetragen sind. Das Wort mit den meisten Dokument-Einträgen ist bei uns „Clinton“ mit 216 Einträgen. Die exakten Zahlen hängen von Eurer Art der Tokenisierung und ein paar anderen Variablen ab; die ungefähre Größenordnung sollte aber mit diesen Angaben übereinstimmen.

Die Indexierung *dieser einen* Datei sollte nicht länger dauern als 10 Minuten (eher deutlich weniger).

# 2 Programmieren der Suchfunktionalität

Die Suchanfrage soll die boolschen Verknüpfungen UND, ODER und NICHT („-“) erlauben. Für die Suchanfrage „Bosnia AND -war“ sollen zum Beispiel alle (bzw. die n relevantesten) Dokumente geliefert werden, die das Wort „Bosnia“, aber nicht das Wort „war“ enthalten. Die Ergebnisse sollen alle über den im ersten Teil erstellten Index bestimmt werden, z.B. heißt „-war“ hier *nicht*, dass „war“ nicht im Dokument vorkommt, sondern dass das Dokument nicht unter „war“ im Index steht.

Der Einfachheit halber genügt es, Negationen nur für einzelne Wörter zuzulassen, also Suchanfragen wie „-(war AND Bosnia)“ zu verbieten. Ausserdem genügt es, sich auf ungeklammerte Suchanfragen einzuschränken: Wenn mehrere boolsche Operatoren vorkommen („Bosnia OR Croatia AND -war“) steht es der Implementierung frei, welche Präzedenz (Klammerung) der Operatoren gewählt wird.

## Bonusaufgaben

*Bonusaufgabe 1:* Implementiere Index und Suchfunktionalität so, dass die Dokumente nach Relevanz sortiert ausgegeben werden, d.h. im Prinzip die Dokumente mit den höchsten tf-idf-Werten für die Suchbegriffe zuerst. Für mehrere Suchbegriffe bzw. mit Operatoren verknüpfte Anfragen müsst ihr einen Weg finden, die Werte sinnvoll zu kombinieren.

*Bonusaufgabe 2:* Implementiere eine Suchmöglichkeit für beliebig geklammerte boolsche Ausdrücke, so dass die Präzedenz entsprechend der Klammerung beachtet wird (kniffelig).

---

## Abgabe bis Donnerstag, 13.01.11, 14:00 Uhr