TDT4117 - Øving 4

Ole Christer Selvig, Håkon Løvdal og Kristoffer Andreas Dalby

19. november 2013

Task 1 - Miscellaneous

Task A

Centralized crawler-indexer architecture

Er en sentralisert måte å prosessere spørringer og indekser. I denne arkitekturen blir det brukt en sentral server til å kontrollere webcrawlers. Arkitekturen er enkel å vedlikeholde og håndtere ettersom det er lite elementer involvert i denne prosessen. Det er dessverre andre problemer med arkiteturen, som single point of failure. Et målrettet angrep mot en server kan for eksempel ta ned hele systemet.

Distributed crawler-indexer architecture

En distribuert arkitektur bruker i motsettning til sentralisert mange noder til å fordele arbeidsmengden og kan takle problemene på en annen måte fra flere steder. Det er ikke lenger et problem om en node skulle gå ned, men det er vanskligere å faktisk håndtere og organisere systemet.

Task B

Document preprocessing

Stegene i Dokument preprossessering er:

Steg 1 - Lexical analyse

Lexical analyse går ut på å behandle mellomrom, nummer og tegn i en tekst, med formål om å gjøre en samling av karakterer til samlinger med ord. For eksempel skal "state-of-the art" og "state of the art" behandles på samme måte da det har samme betydning.

Steg 2 - Eliminere stoppord

Eliminasjon av stoppord går ut på å fjerne ord som kan ødelegge indekseringen fordi de er ubetydlige i indekserings sammenheng og forekommer for mange ganger. Eksempler på stoppord i norsk som ofte blir fjernet er: i, og, å, er.

Steg 3 - Stemming

Stemmning handler om å behandle ord hvor man har et ord som er stammen til en rekke andre ord, gjerne hvor endelsen er annerledes. Et eksempel på dette er ordet tilkoble som er stammen til, tilkoblinger, tilkobler, tilkoblingene.

Steg 4 - Valg av indeks termer

Valg av indekstermer handler som navnet tilsier at man skal velge hvilke termer som skal brukes. Man må gjerne velge hvor mange termer man skal bruke og hvor spesifikke disse skal være. Dette bestemmer hvor dyp indekseringen skal være. Her er det vanlig å gruppere subjektiv og behandle subjektiv som forekommer nær hverandre som et enkelt element. Som foreksempel computer science.

Steg 5 - Konstruksjon av term kategoriserte strukturer

I dette steget skal man strukturere gruppene og termene man valgte i forrige steg. En måte å organisere dette er i en tesaurus. En tesaurus er en struktur på et kontrollert vokabular der termene står i relasjon til hverandre.

Task 2 - Lucene

Oppgave a

Kode lagt til i *MyDocument.java*.

```
public class MyDocument {

        public static Document Document(File f)
                        throws java.io.FileNotFoundException {

                // make a new, empty document
                Document doc = new Document();

                // use the news document wrapper
                NewsDocument newsDocument = new NewsDocument(f);

                //TODO create structured lucene document

                //============START OUR CODE
                    ======================//
                // Adding path
                doc.add(new Field("path", f.getAbsolutePath(),
                    Store.YES, Field.Index.ANALYZED));

                // Adding from
                doc.add(new Field("from", newsDocument.getFrom(),
                    Store.YES, Field.Index.NO));

                // Adding subject
                doc.add(new Field("subject", newsDocument.
                    getSubject(), Store.YES, Field.Index.ANALYZED))
                    ;

                // Adding content
                doc.add(new Field("content", newsDocument.
                    getContent(), Store.YES, Field.Index.ANALYZED))
                    ;
                //============END OUR CODE======================//

                // return the document
                return doc;
        }

}
```

Kode endret i *MyIndexFiles.java*.

```
        public static void main(String[] args) {
                String usage = "java␣org.apache.lucene.demo.
                    IndexFiles␣<root_directory>";

                //========CHANGED SOME CODE HERE
                    =========================//
                String pathToNews = "";
                if (args.length == 0) {
                        System.err.println("Usage:␣" + usage);
                        pathToNews = "/Users/hakloev/git/TDT4117/
                            oving4/20news-part";
                        //System.exit(1);
                } else {
                        pathToNews = args[0];
```

```
                }
                //========END======ALSO ADDED pathToNews IN LINE
                    41======//

                if (INDEX_DIR.exists()) {
                        System.out.println("Cannot␣save␣index␣to␣'"
                             + INDEX_DIR
                                        + "'␣directory,␣please␣
                                            delete␣it␣first");
                        System.exit(1);
                }

                final File docDir = new File(pathToNews);  // <-
                    HERE
```

Output fra kommandovinduet ved kjøring av *MyIndexFiles.main()* etter modifisering av kode.

```
Usage: java org.apache.lucene.demo.IndexFiles <root_directory>
Indexing to directory 'index'...
adding /Users/hakloev/git/TDT4117/oving4/20news-part/40008
adding /Users/hakloev/git/TDT4117/oving4/20news-part/40027
adding /Users/hakloev/git/TDT4117/oving4/20news-part/40062
...
for alle 1907 dokumentene
...
adding /Users/hakloev/git/TDT4117/oving4/20news-part/59648
adding /Users/hakloev/git/TDT4117/oving4/20news-part/59652
Optimizing...
1811 total milliseconds

Process finished with exit code 0
```

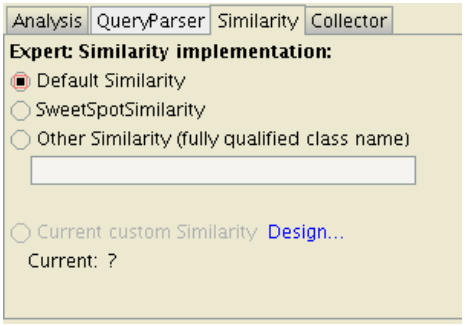
Oppgave b

Forklaring av systemet

Luke er et program som bruker indeksene vi genererte med Lucene i forrige oppgave til å vise og modifisere innhold på flere måter.[1] I denne øvingen har vi vært interessert i å sortere etter fire felter: *content, from, path* og *subject*. Ved å sørge for at Lucene indekserer etter disse feltene, kunne vi generere indeksfiler som kunne brukes i Luke. Vi kunne da søke etter gitte termer i de ulike feltene av dokumentet.

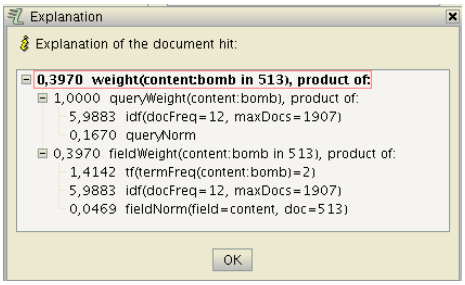
Figur 1: QueryParser

Lucene benytter seg av den boolske-modellen til å finne termer i et dokument, og rangerer de med vektor-modellen. Ser vi på figur 1 ser vi at QueryParser er den boolske-modellen og at klausulen er satt til logisk-ELLER. Figur 2 viser at similaritetsmodellen er satt til vektor-modellen, da den er "Default Similarity"i Lucene. Det skal legges til at cosinusformelen som brukes er modifisert i forhold til den vi har lært faget, men prinsippet er det samme.[2]



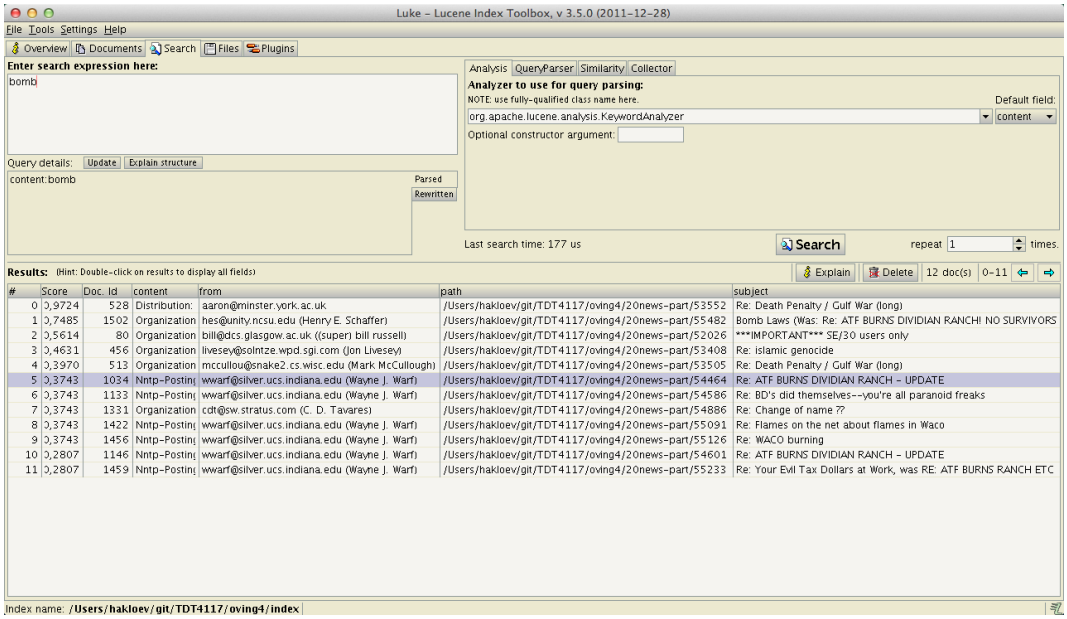
Figur 2: Similaritetsmodell

I figur 3 kan vi se tallene som er til grunn for utregningen av similariteten til dokument 513. Vi ser dokumentet bli rangert til 0,3970, noe som gir plass fem i rangeringen. Dette er fordi rangeringen er produktet av en serie tall, nærmere bestemt produktet mellom spørringsvekt og feltvekt (*query weight* og *field weight*). Her er spørringsvekt igjen et produkt av *idf* (inverse document frequency) og en konstant *queryNorm*. Feltvekt er et produkt av termfrekvens, idf og fieldnorm. Fieldnorm blir regnet ut som følge av lengden på feltet etter stemming, tokanisering og andre operasjoner på dokumentet.

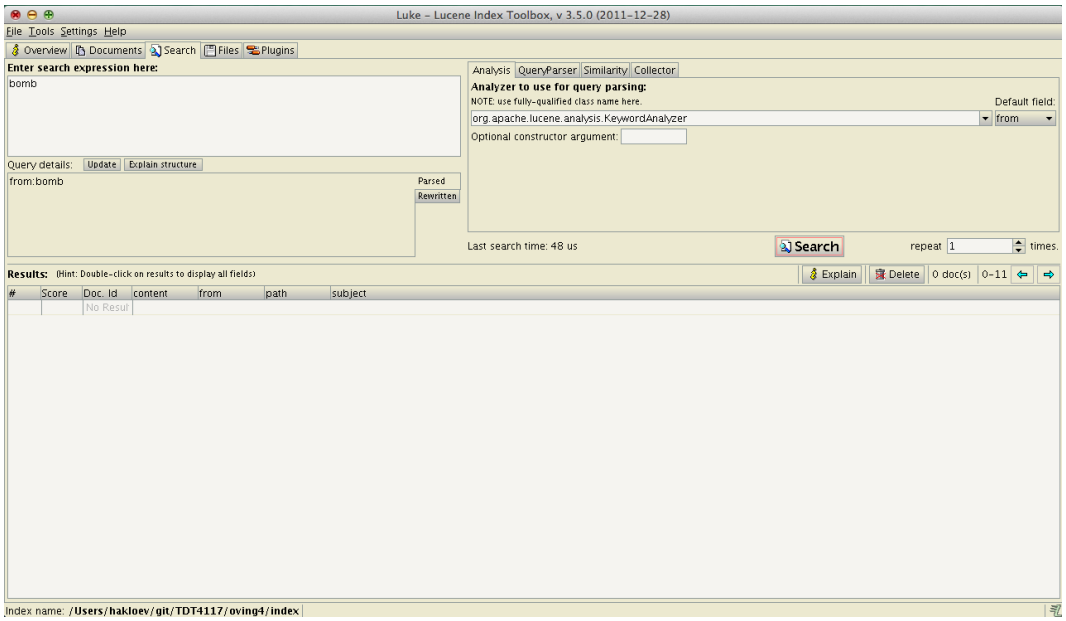


Figur 3: Data for dokument 513

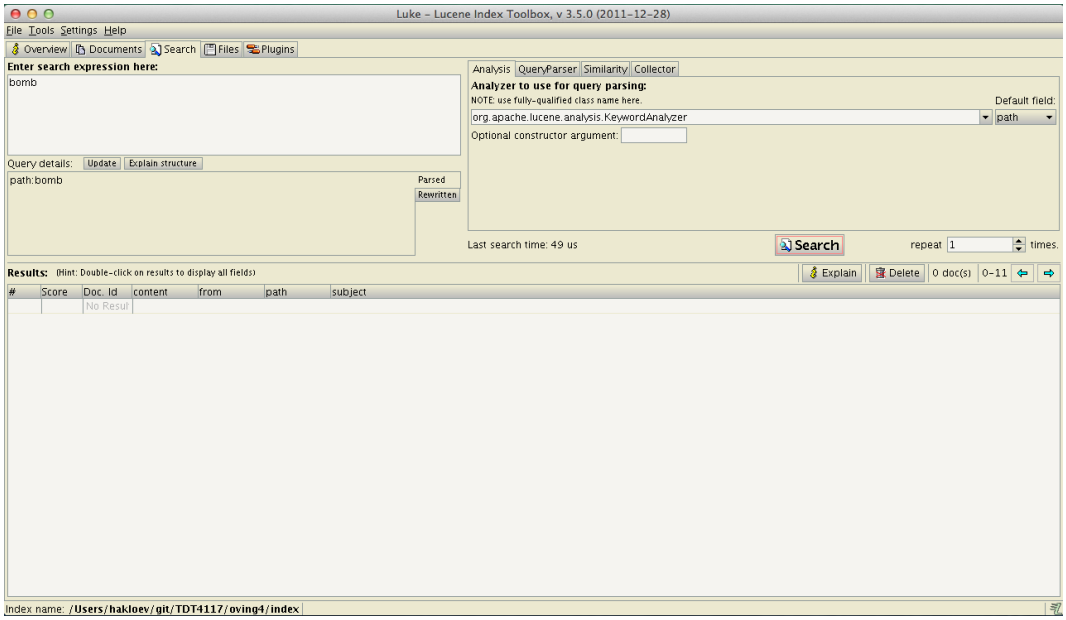
[1] https://code.google.com/p/luke/
[2] http://lucene.apache.org/core/3_0_3/api/core/org/apache/lucene/search/Similarity.html



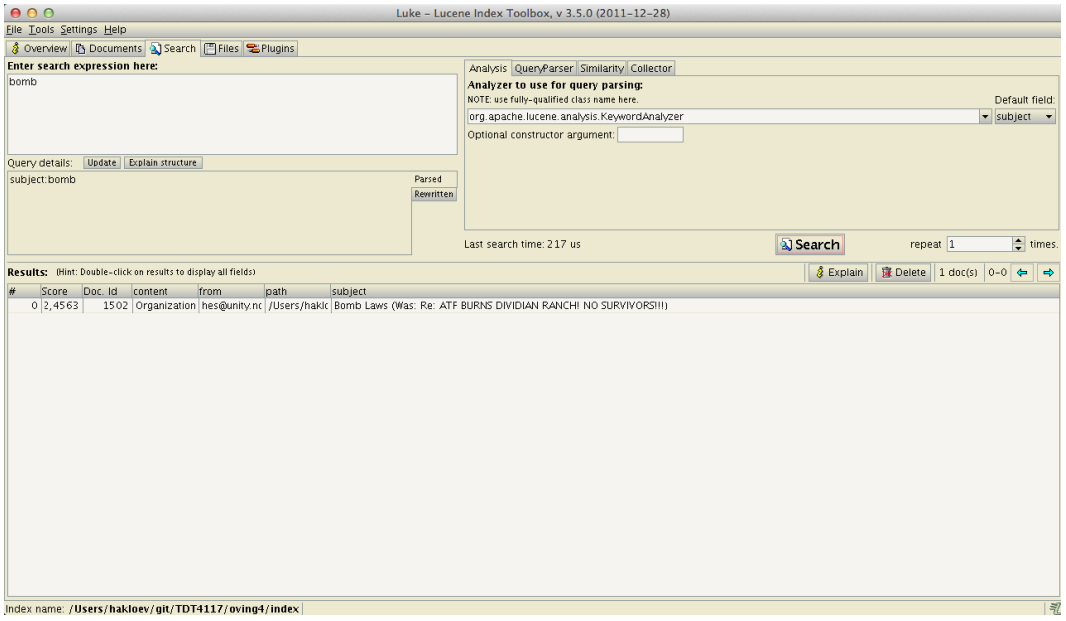
Figur 4: Søk etter bomb i content



Figur 5: Søk etter bomb i from



Figur 6: Søk etter bomb i path



Figur 7: Søk etter bomb i subject