Zürcher Hochschule für Angewandte Wissenschaften

Seminar Information Retrieval

Evaluierung der Retrieval-Leistung einer Search Engine am Beispiel einer privaten MP3-Sammlung

Author:
Philipp Schalcher

Betreuer:
Ruxandra Domenig

7. Mai 2014

Danksagung

Inhaltsverzeichnis

Zusammenfassung

Kapitel 1

Einleitung

In der heutigen Zeit wird der Mensch von einer Fülle an Informationen überflutet. Würde er nicht gewisse Eindrücke selber filtern, könnte das zu einem Kollaps führen. Der Mensch hat das Glück, solche Dinge von der Natur eingebaut zu haben. Im Gegensatz zum Menschen besitzen Informationssysteme keine integrierten Filter. Das beste Beispiel hierfür ist Google. Es gibt eine riesige Menge an Daten, die der Suchmaschine ihr Wissen verleiht.

Lucene ist eine Bibliothek, welche in verschiedene Projekte eingebaut werden kann, um so eine mächtige Suchmaschine auf Basis von Indexen zu bekommen. Lucene enthält alle relevanten Funktionen, die benötigt werden, um Informationen zu durchsuchen. Hier liegt die Herausforderung, eine Suchmaschine für ID3-Tags von MP3-Dateien zu bauen, da Lucene hauptsächlich für Textdateien (PDF,TXT,DOCX,eBooks,usw.) genutzt wird. Für MP3-Dateien stehen andere Probleme an (Wie extrahiere ich die ID3-Tags aus einer MP3-Datei).

Diese Arbeit soll die Information Retrieval Leistung der Suchengine Lucene analysieren. Darin enthalten sind eine Programmierung einer kleinen Suchmaschine, die MP3-Dateien innerhalb eines Ordners indexiert und danach durchsucht. Dabei beschränke ich mich in der praktischen Arbeit auf das indexieren der ID3-Tags. Diese Arbeit soll nicht als Anleitung zur Erstellung einer Suchmaschine dienen!

Da Lucene natürlich auch den Inhalt einer Datei analysiert, muss diese Arbeit ein bisschen angepasst werden. Da MP3-Dateien keinen Text als Inhalt haben, möchte ich daher nur theoretisch aufzeigen, wie anhand von Teilen eines Liedes das entsprechende Lied gesucht werden kann. Dies zu programmieren sprengt den Rahmen der Arbeit, somit werde ich am Beispiel von Shazam nur eine theoretische Lösung aufzeigen.

Kapitel 2

Hauptteil

Lucene

Was ist Lucene? Dies ist die erste Frage, die ich mir zu Beginn der Arbeit gestellt habe. Da mir das Produkt gänzlich unbekannt war, galt es zuerst Informationen zu sammeln.

Apache Lucene eine Suchengine, die sich auf Text und eine hohe Performance spezialisiert hat. Dabei ist die Engine mittlerweile in verschiedene Sprachen übersetzt worden. Der Apache Lucene Core ist der Hauptteil der Software und ist in Java geschrieben. Das beste an Lucene ist wohl, dass es gratis zur Verfügung steht. Somit kann jeder Entwickler eine mächtige Suchmaschine in seine Programme einbauen.

Bei einer Suchmaschine liegen die Stärken im Resultat welches geliefert wird. Lucene bietet auch hier wieder einige Funktionen, die das Endergebnis schnell und korrekt ergeben sollen. Dazu gehören:

- Ranked Searching - Die besten Resultate werden als Erste zurückgegeben.
- Verschiedene Query-Typen
- Feldsuche (Hier Titel, Album, Künstler, Jahr, Songtext).
- Mehrfache Indexe durchsuchen mit zusammengefasstem Ergebnis.
- Schnell
- Speichereffizient
- Tippfehler-tolerant

Mit diesen und weiteren Gimmicks wird Lucene auf der Webseite `lucene.apache.org/core/` angepriesen. Für meine Arbeit habe ich die Bibliothek in der Version 3.6.2 verwendet, da meine Quelle ebenfalls mit einer 3er-Version gearbeitet hat.

Information Retrieval

> The IR Problem: The primary goal of an IR system is to retrieve all the documents that are relevant to a user query while retrieving as few non-relevant documents as possible. - Buch: Modern Information Retrieval

Dieses Zitat bezeichnet sehr gut um was es bei Information Retrieval geht. Die Menschheit speichert seit 5000 Jahren Informationen in verschiedenen Systemen, um danach über Indexe oder andere Suchmechanismen an wichtige Informationen zu kommen. Im einfachsten Fall möchte ein User nach einer Suche einen Link zu einer Webseite von einer Organisation, Firma oder sonstigen Quelle. Information Retrieval alleine dreht sich nicht nur um Suchmaschinen. Die Definition aus dem Buch Modern Information Retrieval lautet wie folgt:

> Information retrieval deals with the representation, storage, organization of, and access to information items such as documents, Web pages, online catalogs, structured and semi-structured records, multimedia objects. The representation and organization of the information items should be such as to provide the users with easy access to information of their interest. - Buch: Modern Information Retrieval

Zu Beginn war Information Retrieval nur eine Kategorie, die für Bibliothekare und Informationsexperten interessant war. Dieser Umstand änderte sich aber schlagartig, als das Internet aufkam. Informationen waren nun zugänglich und konnten von fast jedem Menschen abgerufen werden. Mit dem Internet wurde es wichtiger, gute Ergebnisse beim Suchen nach Informationen zu bekommen. Information Retrieval hatte die breite Masse erreicht.

Ein Grundproblem bei IR-Systemen ist, welche Informationen für die gewünschte Suche überhaupt relevant sind. Wie soll die Software entscheiden, welche Informationen am besten auf die Anforderungen des Benutzers zutreffen? Dabei gibt es verschiedene Möglichkeiten. Ein Beispiel ist das sogenannte Ranking. Dabei wird einem Dokument, welches häufiger erscheint, ein höheres Ranking gegeben. Somit weiss die Suchmaschine, dass es wahrscheinlicher ist, dass in diesem Dokument gesuchte Informationen zu finden sind, die mit den Anforderungen übereinstimmen.
Eine weitere Möglichkeit besteht in der örtlichen Ablage von Informationen. So können die "am nächsten liegenden"Informationen auch die richtigen sein. Suche ich zum Beispiel in Google nach einem Detailhändler, möchte ich natürlich die Händler im Ergebnis erhalten, die am nächsten zu mir liegen.
Als Drittes ist die Grösse eines Dokumentes ebenfalls eine Variable. So können kleine Dokumente bevorzugt werden, da sie schnell heruntergeladen und veranschaulicht werden können.

Information Retrieval Systeme müssen komplexe Anforderungen erfüllen. Mit den oben genannten drei Beispielen für Ergebnissen, folgt der Schluss, dass IR-Systeme nie die eine perfekte Lösung anbieten können. Die Anfragen sind zu komplex um alle möglichen Varianten in eine Suchmaschine zu integrieren.

Lucene ist genau solch ein IR-System. Dieses System soll nun analysiert werden. Darunter fällt die Analyse der verschiedenen Analyzer. Konkret handelt es sich um den Standard Analyzer, den Whitespace Analyzer, den Stop Analyzer und den Simple Analyzer. Hier möchte ich mich darauf konzentrieren, wie genau die Informationen in den Index gespeichert werden. Ein ebenfalls wichtiger Faktor ist die Zeit. Einerseits die Dauer der Indexerstellung und danach die Dauer der Suche mit den verschiedenen Einstellungen. Als Letztes möchte ich mich um die Analyse des Ergebnisses kümmern, wobei ich eine Testmenge an MP3 benutze um damit die tatsächlichen Suchergebnisse mit der erwarteten Menge zu vergleichen.

Lucene Search Engine

Das entwickelte Programm hat den Namen "Lucene Search Engine"und wurde in Java entwickelt. Im Programm kann unter dem Punkt "Pfad"der Ablageort der MP3-Dateien angegeben werden. Mit dem Button Index wird dann in diesem Pfad ein Ordner Index erstellt, in welchem der Lucene-Index erstellt wird. Dabei durchsucht das Programm alle MP3-Dateien im angegeben Pfad, auch in Unterordnern. Aus diesen Dateien werden dann die Informationen Titel, Album, Künstler, Jahr, Songtext, Pfad und Dateiname extrahiert und in den Index geschrieben. Gesucht wird dann in den Feldern Titel, Album, Künstler, Jahr, Songtext und Dateiname. Als Ergebnis wird der Pfad zur entsprechenden Dateien ausgegeben.

Das GUI

In diesem Abschnitt möchte ich ganz kurz die grafische Benutzeroberfläche erläutern. Ich gehe nicht weiter auf die Programmierung ein, da dies nur ein kleiner, eher unwichtiger Teil der Search Engine ist. Zuerst betrachten wir die Menüleiste. Unter dem Punkt Datei

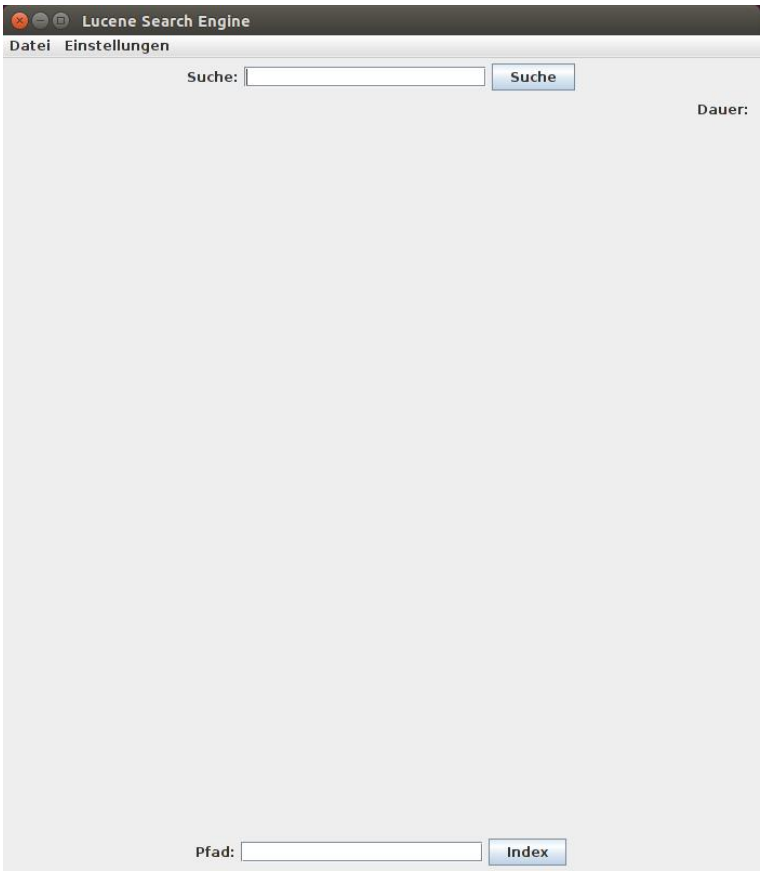

Abbildung 2.1: GUI des Programmes[1]

ist nicht weiteres zu finden als ein Punkt "Schliessen"der das Programm beendet. Interessanter ist da der Punkt Einstellungen. Öffnet man diesen, erhält man eine Auswahl der 4 erwähnten Analyzer. Je nach Auswahl wird der Index auf andere Weise erstellt und die Suche durchgeführt. Es kann jeweils nur ein Analyzer ausgewählt werden. Falls man ihn ändert sollte man auch den Index neu erstellen.

Indexieren

Am unteren Rand der Applikation befindet sich das Feld Pfad. Dort wird der Pfad zu den abgelegten Dateien eingefügt. Dies funktioniert für Linux wie für Windows und Mac. Sobald dies erledigt wurde, kann mit dem Knopf die Indexierung gestartet werden. Als

[1] Quelle: Screenshot

Analyzer wird die Auswahl aus den Einstellungen genommen. Der Index wird im gleichen Pfad erstellt und in einem Unterordner mit dem Namen "Index"abgelegt. Dies geschieht automatisch und kann nicht geändert werden. Sobald die Indexierung beendet ist, wird auf der rechten Seite die Dauer in Millisekunden angezeigt. Je mehr MP3-Dateien indexiert werden müssen, desto länger dauert der Vorgang. Nach der Indexierung ist die Suchma-

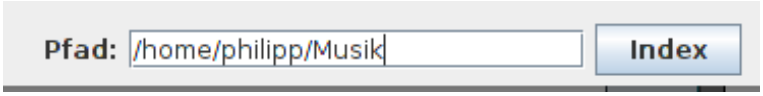

Abbildung 2.2: Beispiel eines angegebenen Pfades in Ubuntu[2]

schine bereit und kann verwendet werden. Das praktische an dieser Lösung liegt darin, dass die MP3-Dateien zum Beispiel auch auf einem externen Speichermedium liegen können und der Index auf diesem Medium erstellt werden kann (gilt natürlich nicht für CD/DVDs). So kann theoretisch für jeden Ablageort eine eigener Index erstellt werden. Genaueres zur Indexierung und wie sie gelöst wurde ist im Abschnit "Der Indexer"zu lesen.

Suchen

Als Erstes muss gesagt werden, dass für die Suche das Pfadfeld weiterhin den Eintrag enthalten muss. Dies ist wichtig, da sonst die Engine nicht weiss, wo sich der Index befindet. Am oberen Rand befindet sich das Suchfeld. Der Benutzer kann hier seine Suchbegriffe eingeben, welche danach im Index gesucht werden. Wieder aktiviert man mit einem Klick auf Suche die Aktion. Sobald die Engine die Ergebnisse bekommen hat, wird die Dauer der Suche am rechten Rand angezeigt. Unterhalb des Suchfeldes wird eine Tabelle ausgegeben, welche die gefundenen Resultate beinhaltet. Lucene bietet integrierte Funktionen, um

Abbildung 2.3: Beispiel eines eingegebenen Suchbegriffes[3]

die Suche besser auf seine Wünsche anzupassen. Hier ist eine Liste, wie man mit der Suchmaschine detaillierter suchen kann:

- Eingabe: song1
  Sucht in den indexierten Feldern nach diesem Begriff
- Eingabe: song1 song2
  Sucht in den indexierten Feldern nach den beiden Begriffen. Gibt Resultate aus, welche Begriff 1 oder Begriff 2 oder beide beinhalten.

[2]Quelle: Screenshot
[3]Quelle: Screenshot

- Eingabe +song1 +song2
  Sucht in den indexierten Feldern nach den beiden Begriffen. Gibt Resultate aus, welche beide Begriffe beinhalten.

- Titel:song1
  Sucht in dem indexierten Feld Titel nach dem Begriff.

- Titel:song1 -Album:album1
  Sucht in den indexierten Feldern nach dem Begriff. Gibt als Resultat die Dateien zurück, welche den Titel besitzen aber nicht zum Album gehören.

- (song1 OR song2) AND album1
  Sucht in den indexierten Feldern nach den Begriffen. Gibt als Resultat die Dateien zurück welche song1 oder song2 im besitzen und in album1 zu finden sind.

- "song1"
  Sucht in den indexierten Feldern nach exakt diesem Wort oder Text.

- song1*
  Sucht in den indexierten Feldern nach Werten, die mit song1 beginnen.

- song1∼
  Sucht in den indexierten Feldern nach Werten, die ähnlich wie song1 sind.

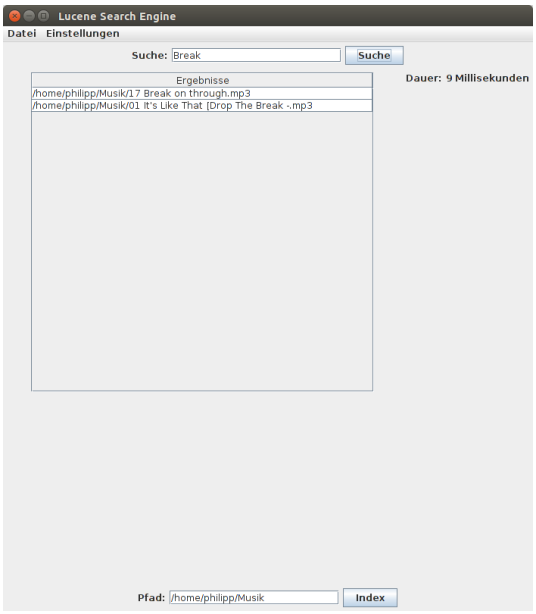

Abbildung 2.4: Beispiel gesuchter Begriff[4]

[4]Quelle: Screenshot

Der Indexer

In diesem Teil möchte ich kurz auf die Klasse Index eingehen, welche nach den MP3-Dateien sucht und die Informationen aus den ID3-Tags zieht. Die Klasse Index wird durch das GUI gestartet, indem man den Kopf "Index"betätigt. Als Parameter erhält er den Inhalt des Textfeldes "Pfad"und welcher Analyzer genutzt werden soll. Nun wird der Pfad für den Index definiert. Dieser liegt immer im angegeben Verzeichnis im Unterordner "Index". Danach initialisiert die Klasse den Indexer und erstellt den Analyzer. Im Anschluss startet er die Messung der Zeit und beginnt die Dateien zu suchen und die Informationen zu extrahieren. Dies geschieht in der Methode index(String dataDir, FileFilter filter). Falls er einen Ordner findet, öffnet die Klasse den Ordner und sucht weiter nach Dateien.

Hat die Methode eine MP3-Datei gefunden, wird sie mit der Methode indexFile(File f) in ein Document-Typ gewandelt, welcher die Informationen aus den ID3-Tags enthält. Per writer.addDocument(doc) wird das MP3 in den Index aufgenommen. Die ID3-Tags werden in der Methode getDocument(File f) einzeln aus jedem MP3 ausgelesen. Dabei standen verschiedene Lösungen zur Verfügung. Dazu aber später mehr im Kapitel "ID3-Tags".

Sobald alle MP3-Dateien aufgenommen wurde, berechnet die Klasse als Letztes, wie lange der Vorgang gedauert hat. Dieses Ergebnis wird dann, wie schon erwähnt, auf dem GUI ausgegeben. Der Index wurde erstellt und kann nun zur Suche genutzt werden.

Der Searcher

Ein Index alleine nützt uns nichts, wenn nicht ein Searcher implementiert wird, der diesen Index auch durchsuchen kann. Dazu ist in diesem Projekt die Klasse "Searcher"vorhanden. Wie schon beim Indexer, erhält der Aufruf per Knopfdruck auf den Button "Suche"die Parameter Pfad, Analyzerart und die Anfrage. Der Pfad wird direkt aus dem Indexpfad gelesen. So findet das Programm den korrekten Index. Danach startet die Methode search(String indexDir, String q, String analyzerArt).

Die Methode search öffnet den vorgegebenen Pfad um den Index zu lesen. Danach wird ein Parser erstellt, der mehrere Felder absuchen kann (MultiFieldQueryParser). Je nach Analyzerart wird eine andere Version erstellt. Dieser Parser wandelt die Benutzeranfrage (hier q) so um, dass Lucene die Anfrage versteht. Die Methode durchsucht den Index und ergibt maximal 100 Ergebnisse. Die einzelnen gefundenen Inforationen werden wieder in den Typ Document umgewandelt. Zum Schluss wird eine ArrayList mit Document zurückgegeben, welche alle Ergebnisse beinhaltet. Die Liste wird jedesmal bei der Suche gelöscht und wieder von neuem befüllt. Der Inhalt der ArrayList wird zum Schluss im GUI ausgegeben, sowie auch die Dauer der Suche.

ID3-Tags

ID3-Tags sind Informationen, welche direkt in die Dateien eingebettet wurden. Sie beinhalten wertvolle Informationen wie Titel, Künstler, Album, Jahr, Genre, usw. ID3 hat sich mittlerweile zum Standard für MP3 entwickelt und wird von Programmen wie iTunes, Windows Media Player und VLC genutzt. Leider können nicht alle Programme mit jeder ID3-Version umgehen. Dies kann zu Komplikationen führen.

Um in meiner Applikation diese Daten zu extrahieren, standen drei Möglichkeiten zur Auswahl. Diese waren manuelles extrahieren in Java, benutzen einer Bibliothek und einsetzen von Tika.

Manuelles extrahieren

Das auslesen von ID3v1 ist relativ einfach. Die Informationen sind immer fest in einem Block von 128 Bytes am Ende der Datei gespeichert. Bei Offset 3 mit einer Länge von 30 wird zum Beispiel der Songtitel gespeichert. Folgende Tabelle gibt die Felder und deren Position wieder:

Offset	**Länge in Bytes**	**Bedeutung**
0	3	Kennung eines ID3v1-Blocks
3	30	Songtitel
33	30	Künstler
63	30	Album
93	4	Jahr
97	30	Kommentar
127	1	Genres

Diese können so direkt angesprochen werden. Schwieriger wird es mit der Version 2. Ab Version zwei wird im Header der Datei angegeben, ob es sich um ID3 der Version 2 handelt. Die Tags werden dann so kodiert, dass ein Player sie nicht versteht und als fehlerhafte Informationen interpretiert. So werden die Teile der Datei übersprungen und nicht abgespielt. Neu können auch Bilder in diese Feldern abgelegt werden. Allerdings ist die Implementation sehr mühsam.

Da diese Lösung viel Zeit verschlingen würde, da für jede Version der ID3-Tags auch eine Methode geschrieben werden müsste, die die entsprechenden Tags ausliest, habe ich diese Lösung nicht in Betracht gezogen. Ausserdem muss ich das Rad nicht neu erfinden, wenn schon Möglichkeiten vorhanden sind, welche meine Anforderungen erfüllen. So kommen wir zum nächsten Kapitel.

Tika

Tika ist ein Framework, das im Oktober 2008 zur Lucene-Familie hinzugefügt wurde. Es besitzt die gleichen Standard-APIs wie Lucene. Tika selber ist nur eine Ergänzung, welche verschiedene Parser für Dateien beinhaltet. Darunter auch MP3. Allerdings können laut

Buch “Lucene in Action”nur ID3v1-Tags ausgelesen werden. Dies erschwert unser vorhaben wieder, da verschiedene Versionen eingesetzt werden können. Ausserdem muss Tika auf andere Open-Source Projekte aufsetzen, um Dateien zu suchen und zu öffnen. Dies kann das Framework nicht selbst tätigen.

Aus diesen Gründen konnte ich Tika aus meinen Möglichkeiten streichen, da so der Aufbau der Applikation nur komplizierter wurde. Und mit der Einschränkung auf ID3v1 war Tika nicht wirklich eine gute Wahl für mein Vorhaben. Ausserdem exportiert Tika die Informationen in eine XHTML-Datei. Das heisst, dass ich zusätzlich zum Index nochmals Dateien ablegen müsste. Somit blieb mir die letzte Lösung.

Extraktion mit Library

Im Internet gibt es verschiedene Bibliotheken, die frei zur Verfügung stehen und sich um solche Dinge, wie die Extraktion von ID3-Tags, kümmern. Meine Wahl fiel auf die Lösung JAudioTagger. Dies ist ein kleines JAR-File, welches problemlos in das Projekt integriert werden konnte. Mit wenigen Code-Zeilen ermöglichte mir die Klasse, dass die Informationen aus den MP3-Dateien in den Index geschrieben werden konnten.

Mit 3 Zeilen wird der Vorgang initialisiert.

```
AudioFile audioFile = AudioFileIO.read(f);
Tag tag = audioFile.getTag();
AudioHeader header = audioFile.getAudioHeader();
```

Zeile 1 liest die Datei ein und wandelt sie in den Typ AudioFile. Durch diesen Typ können nun mit der Methode getTag() in Zeile 2 die ID3-Tags ausgelesen werden. Als Zusatz (aber hier nicht notwendig) wird der Header für weitere Funktionen aus dem AudioFile ausgelesen. Um nun die bestimmten Tags zu bekommen braucht es nur noch den folgenden Befehl:

```
tag.getFirst(FieldKey.TITLE)
```

Mit diesem Befehl wird der spezifische Eintrag bei Titel gelesen. Diese kann dann direkt beim Indexer in ein Feld für den Typ Document gespeichert werden. Natürlich kann anstatt TITLE auch ALBUM, YEAR, ARTIST, usw. genutzt werden. Somit habe ich alle relevanten Informationen aus den Dateien extrahiert und in meinen Index geschrieben.

Analyse der Leistung

Die Suchleistung der Engine möchte ich über zwei Werte vergleichen. Einerseits wäre das natürlich die Zeit, welche benötigt wird, um zu indexieren und zu suchen. Andererseits muss auch das Ergebnis überprüft werden, ob auch die erwarteten Resultate zurückgegeben werden. Die Algorithmen welche verwendet werden, sind bei Lucene die Analyzer. Daher werde ich die vier gezeigten Analyzer überprüfen und danach die Werte vergleichen und ein Fazit ziehen.

Vorbereitung

Um eine korrekte Messung zu bekommen, müssen für alle Analyzer die gleichen Umstände herrschen. Das heisst, dass auf einem USB-Stick eine Auswahl an Dateien gespeichert sind. Diese werden indexiert (ergibt ersten Wert). Dann wird dieser Index (der danach ebenfalls auf dem USB-Stick liegt) wird mit bestimmtem Suchabfragen gefüttert, um dann die Ergebnisse und die benötigte Zeit zu bekommen. Die Tests werden unter Windows 8.1 durchgeführt, auf einem Notebook mit einer 256GB Solid State Disk. Der USB-Stick besitzt 8GB Speicherplatz und ist von der Marke disk2go.com

Testmenge

Für den Test wurden drei Ordner aus der Musikbibliothek ausgewählt. Insgesamt enthält der USB-Stick 960 Dateien und 55 Ordner. Dabei wurden Bilddateien oder ähnliche nicht entfernt. Für die Gegenüberstellung der Suchergebnisse wird das Programm MediaMonkey auf dem Computer installiert und mit der Testmenge gefüttert. Im optimalsten Fall müsste die Search Engine die gleichen Ergebnisse liefern wie das spezifizierte Suchprogramm. MediaMonkey wird in der Version 4.1.1 eingesetzt. Laut MediaMonkey sind auf dem USB-Stick 29 Genres, 257 Interpreten und 51 Alben vorhanden.

Zeitmessung

In diesem Abschnitt möchte ich mich ausschliesslich um die Zeitmessungen bei den verschiedenen Analyzern kümmern. Als erstes beginnen wir mit der Messung der Indexierung. Danach kommt die Messung der Suchanfragen.

Standard Analyzer

Wir beginnen mit dem Standard Analyzer. Zuerst möchte ich erklären, wie dieser bei der Indexierung vorgeht. Man kann den Standard Analyzer nur als umfassend einsetzbar bezeichnen. Er enthält JFlex basierten Grammatik. Somit kann er alphanummerische Werte, Akronyme, Email-Adressen, Webseiten, IP-Adresse, usw. erkennen. Ausserdem besitzt er

die Stopp-Wörter, die auch der Stop-Analyzer verwendet. Bei Stop-Wörtern werden uninteressante Wörter wie Der, Die, Das ausgeblendet und nicht in die Indexierung aufgenommen. Ausserdem ist er der einzige Analyzer, der Strings wie X&Y oder xy@zhaw.ch direkt in den Index übernimmt.

Nun indexieren wir mit dem Standard Analyzer die Dateien auf dem USB-Stick: Wie wir

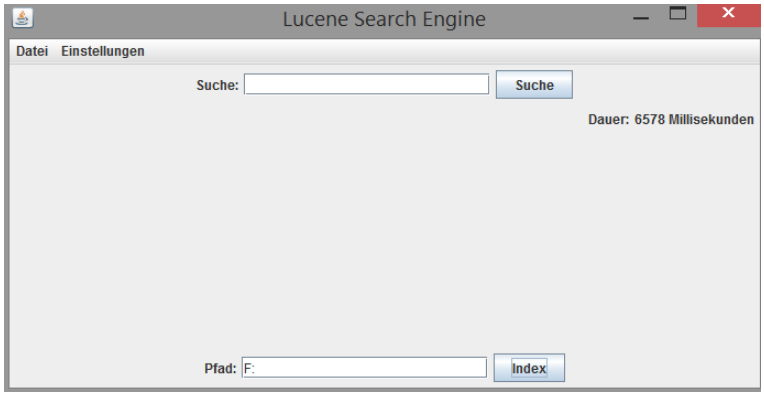

Abbildung 2.5: Indexierung abgeschlossen[5]

auf dem Screenshot sehen brauchte die Indexierung 6578 Millisekunden um den Ordner und den Index auf dem USB-Stick zu erstellen und die Dateien zu durchforsten. Umgerechnet sind das ungefähr 6,6 Sekunden.

Simple Analyzer

Der Simple Analyzer arbeitet auf eine sehr rudimentären Basis. Da Text bei den Analyzern immer in Tokens umgewandelt (also geteilt) werden muss, gehen die Algorithmen auf verschiedene Arten vor. Der Simple Analyzer überprüft den Text nur auf Zeichen die keine Buchstaben sind. Sobald er ein solches Zeichen gefunden hat, teilt er den String an dieser Stelle. Aus dem Beispiel des Standard Analyzers, folgt also das X&Z zu X und Z werden würden und xy@zhaw.ch würde zu xy, zhaw und ch werden. Der Analyzer ist sehr einfach aufgebaut, müsste aber nach meiner Erwartung dafür schneller sein als der Standard Analyzer. Nach meiner Überlegung müsste nur der Whitespace Analyzer noch schneller sein. Später aber mehr dazu.

[5]Quelle: Screenshot

Der Index-Ordner wird auf dem USB-Stick wieder gelöscht und erneut erstellt mit dem Simple Analyzer:

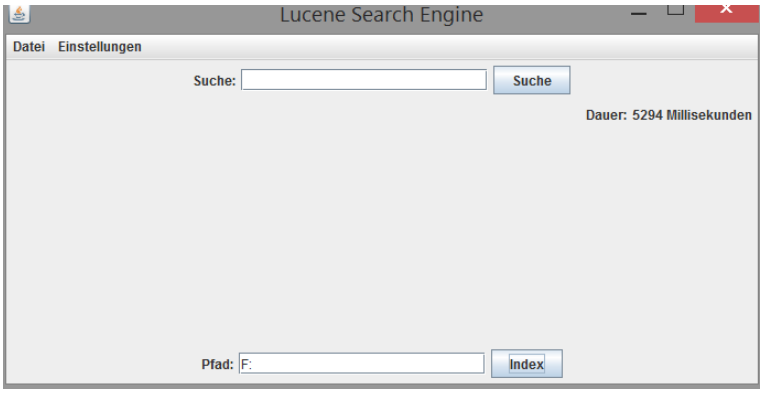

Abbildung 2.6: Indexierung abgeschlossen[6]

Auch hier können wir dem Screenshot wieder ablesen, dass die Indexierung insgesamt 5294 Millisekunden gedauert hat. Die Arbeiten, die das Programm machen musste, waren die Gleichen wie schon beim Standard Analyzer. In Sekdunden umgerechnet dauerte das indexieren als ungefähr 5,3 Sekunden. Wie erwartet war dieser Analyzer schneller als der vorherige.

Whitespace Analyzer

Wie der Simple Analyzer ist der Whitespace Analyzer sehr einfach aufgebaut. Der Text wird in diesem Fall nicht einfach nach Nicht-Buchstaben durchsucht, sondern die Tokens werden aus den Leerschlägen (Whitespaces) im Text erstellt. Wieder am Beispiel von X&Z xy@zawh.ch würde er X&Y und xy@zhaw.ch erkennen. Allerdings hat er, wie der Simple Analyzer das Problem, dass keine Stop-Wörter erkennt werden. Das heisst, unwichtige Wörter wie “und”werden ebenfalls in den Index aufgenommen. Da diese Füllwörter sehr häufig vorkommen, könnten sie im Index eine höhere Gewichtung erhalten, ohne dass sie wirklich Informationen über den Text oder die Datei liefern.

[6]Quelle: Screenshot

Es werden wieder die gleichen Schritte durchgeführt wie zuvor:

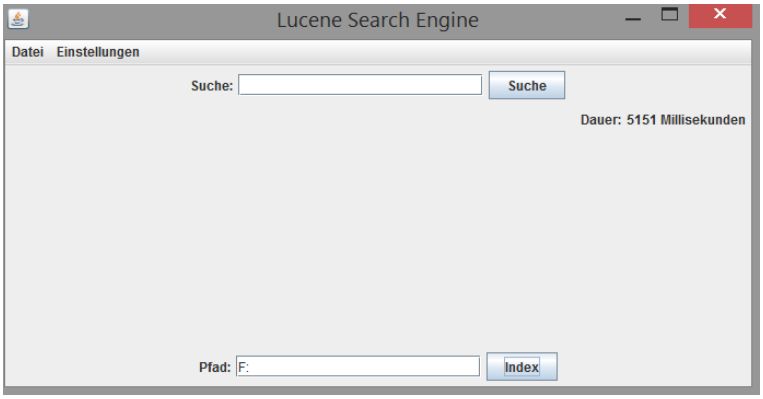

Abbildung 2.7: Indexierung abgeschlossen[7]

Dieses Mal hat der Indexierungsvorgang 5151 Millisekunden gedauert. Wie erwartet war der Whitespace Analyzer noch ein bisschen schneller als der Simple Analyzer. Ich schätze, dass dies aus der Limitierung auf Leerzeichen kommt, da er nicht nach verschiedenen Zeichen suchen muss. Sprich für jedes Zeichen im Text muss nur eine einzige Abfrage gemacht werden. Umgerechnet sind dies ungefähr 5,2 Sekunden.

Stop Analyzer

[7] Quelle: Screenshot