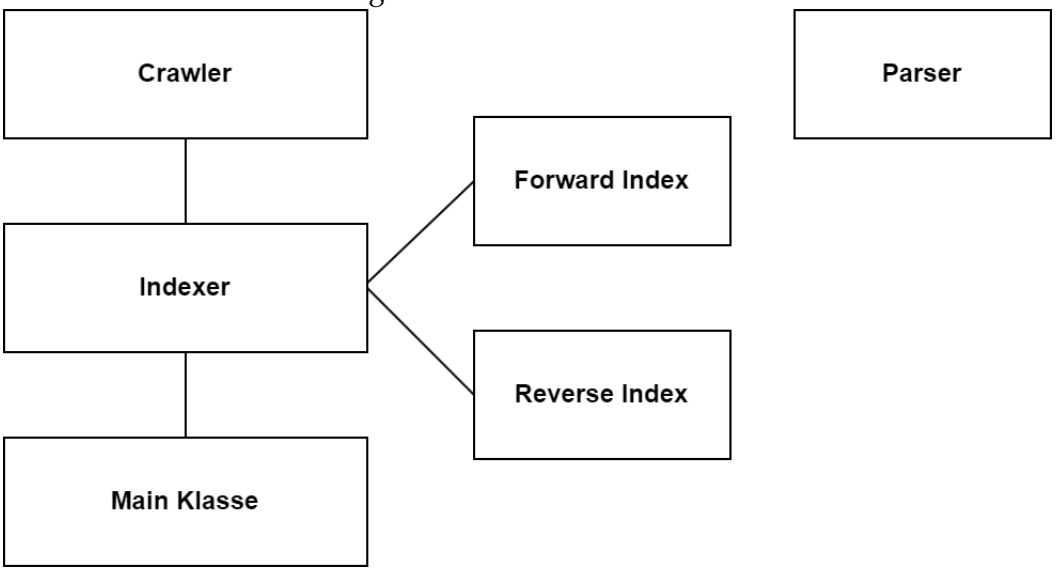
Dokumentation
AnguillaSearch

Robert Bo. Martrlknr

1 Aufbau

Der Aufbau des Programm orientiert sich ganz grob an Abbildung 3.2 aus der Aufgabenstellung.
Eine vereinfachte Darstellung:



- Crawler - Ist für das crawlen der Webseiten zuständig.
- Parser - Eine Utility Klasse, welche gecrawlte Seiten in ein passende Objekte umwandelt und die Tokenizierung, Lemmatisierung etc. übernimmt.
- Indexer - Hier werden die verarbeiteten Objekte in Forward und Reverse Index organisiert und die Suchalgorithmen realisiert.
- Main Klasse - Fungiert als Schnittstelle zwischen dem Endnutzer und den Suchfunktionen des Indexer.

Durch diese Struktur sind die Strukturen Forward Index und Reverse Index als eigene Datenstrukturen vorhanden und können unabhängig vom Indexer erweitert werden. Die Algorithmen werden im Indexer realisiert.

2 Klasse Crawler

Der Crawler ist wesentlicher Teil der Aufgabe 1.
Er nutzt folgende Datenstrukturen um seine Aufgabe zu erfüllen. Eine Queue, in welcher alle URLs, gespeichert werden, welche noch gecrawled werden sollen. Eine Menge, um doppelt Elimination zu garantieren, aller bereits bekannten URLs. Zusätzlich speichert er eine Liste aller Webseiten die gecrawled hat. Wird der Crawler instanziiert erhält er die Seed-URLs und fügte diese in die Queue ein. Anschließend wird eine Schleife durchlaufen, welche die Queue abarbeitet, von jeder Webseite wird mithilfe der Parser Klasse ein Page-Objekt, welches diese repräsentiert. Anschließend kann von diesem Page Objekt die Menge aller Links abgefragt werden. Diese Menge wird mit der Menge der bereits bekannten URLs abgeglichen und ausschließlich neue URLs , werden in die Queue eingefügt. Zum runterladen einzelnen Webseiten wird die Bibliothek JSoup genutzt.
Im Rahmen von Zusatzaufgabe 1 wurde die Option hinzugefügt bei der Instanziierung eine maximale Anzahl an Seiten anzugeben, welche gecrawled werden sollen. Bei jedem Schleifendurchlauf wird diese Bedingung geprüft, ohne Angabe ist die maximale Anzahl an Seiten $2^{31} - 1$.

2.1 Klasse Page

Die Page Klasse wurde erstellt um jede Webseite als eigenes Objekt in einer für unsere Zwecke sinnvollen Art und Weise zu repräsentierten und uns komfortablen Zugriff auf die derzeit benötigten Eigenschaften zu ermöglichen. Bei der Erstellung eines Page Objekt werden bereits die relevanten Eigenschaften wie URL, Titel, Kopf, Inhalt und die Links als Menge gespeichert. Die Links werden als Menge gespeichert, um doppelt vorhandene Links zu eliminieren.
Ebenfalls bei dem Erstellen eines Page Objekts wird eine Liste ohne Stoppwörter der lemmatisierten Token erstellt, dazu nutzt Page unsere Parser Klasse.
Die Klasse stellt somit sicher, dass all diese Daten in brauchbarer Form zur Verfügung stehen. Eine Funktion um Links zu entfernen wurde hinzugefügt um nicht relevante Links für Zusatzaufgabe 1 zu entfernen.

3 Klasse Parser

Bietet eine Funktion zum Erstellen von Page Objekten und hat somit die Aufgabe aus einer heruntergeladenen Webseite die relevanten Daten zu extrahieren.
Erhält die vom Crawler herruntergeladenen Webseiten und trennt URL, Titel, Header, Inhalt und die Menge der Links.
Zusätzlich bietet es noch die Funktion an der Vorverarbeitung an. Hier werden aus einem Text Tokens erstellt, die Stoppwörter entfernt und mithilfe der Stanford CoreNLP Bibliothek Lemmas extrahiert

Die Stoppwortliste wurde von CoreNLP[1] übernommen, da das Projekt angesehen ist und aktiv weiterentwickelt wird. Diese Funktion der Vorverarbeitung wird sowohl in der Page Klasse als auch in der Indexer Klasse verwendet. Die Realisierung dieser Funktionen in der Hilfeklasse Parser, stellt sicher, dass die gleiche Art der Vorverarbeitung in anderen Klassen durchgeführt wird. So werden Suchanfragen, genauso vorverarbeitet wie die eigentlichen Webseiten, was wesentlich ist damit diese gefunden werden können. Änderungen an der Vorverarbeitung können so an einer Stelle im Code erfolgen, diese sind dann auch in der Klasse Indexer und bei den Page umgesetzt.
Die Textnormalisierung hat noch einige Schwachstellen, beispielsweise wird im Netz cheese1 das Token jalape*ñ*os erstellt.

4 Klasse Indexer

Der Indexer enthält die essentielle Suchfunktion und die zugrundeliegenden Rankingalgorithmen und ist somit das Kernstück des Programms.
Zur Realisierung nutzt er Instanzen der Klasse Crawler, Forward Index, Reverse Index und einen Vektor welcher für jedes Token den jeweiligen Inverse Document Frequency Wert enthält. Bei der Instanzierung eines Indexer Objekts, instanziert dieses wiederrum eine Instanz der Klasse Crawler und die relevante Datenbasis zu erhalten.
Die notwendigen Funktionen für die Berechnung von Term Frequency und Inverse Document Frequency Wert, werden ebenfalls in dieser Klasse realisiert. Diese werden als Methoden stehen auch anderen Klassen zur Verfügung. Auch für die Normalisierung eines Vektors auf die Länge 1, bietet diese Klasse eine Funktion. Die Klasse selbst nutzt diese Funktion um den Suchvektor zu normalisieren. Die Berechnung des Vektors mit den IDF Werten ist an dieser Stelle am sinnvollsten, der Indexer verfügt mit dem Reverse Index über eine Liste aller Tokens. Eine Iteration über diese ist somit relativ einfach und garantiert, dass für jedes Token ein Wert errechnet wird. Gleichzeitig benötigt der Forward Index zu Berechnung der TF-IDF Vektoren die IDF Werte. Der Indexer kann dem Forward Index diese IDF Werte somit zur Verfügung stellen.

4.1 Forward Index

Der Forward Index wurde als eigene Klasse realisiert um Änderungen und Erweiterungen an diesem zu erleichtern und so die Wartbarkeit zu erhöhen. Im Rahmen des Praktikums wurde dieser wie in Aufgabe 3 gefordert, erweitert und speichert wie gefordert zu jeder Webseite einen zugehörigen TF-IDF Vektor. Diese Erweiterung wurde als Vererbung realisiert, da es sich hier um eine Spezialisierung des Konzepts Forward Index handelt. Zur Berechnung der TF-IDF Werte nutzt die Klasse die von der Indexer Klasse bereitgestellten Funktionen.
Auch die Normalisierung der TF-IDF Vektoren auf die Länge 1 findet hier statt und auch hierfür werden die Funktionen der Klasse Indexer genutzt.

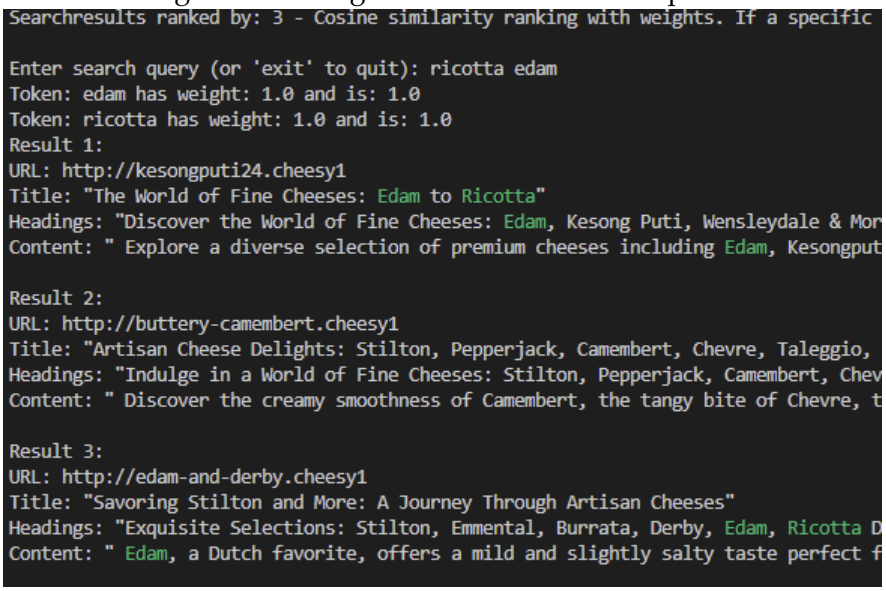
[1] https://github.com/stanfordnlp/CoreNLP/blob/main/src/edu/stanford/nlp/coref/data/WordLists.java

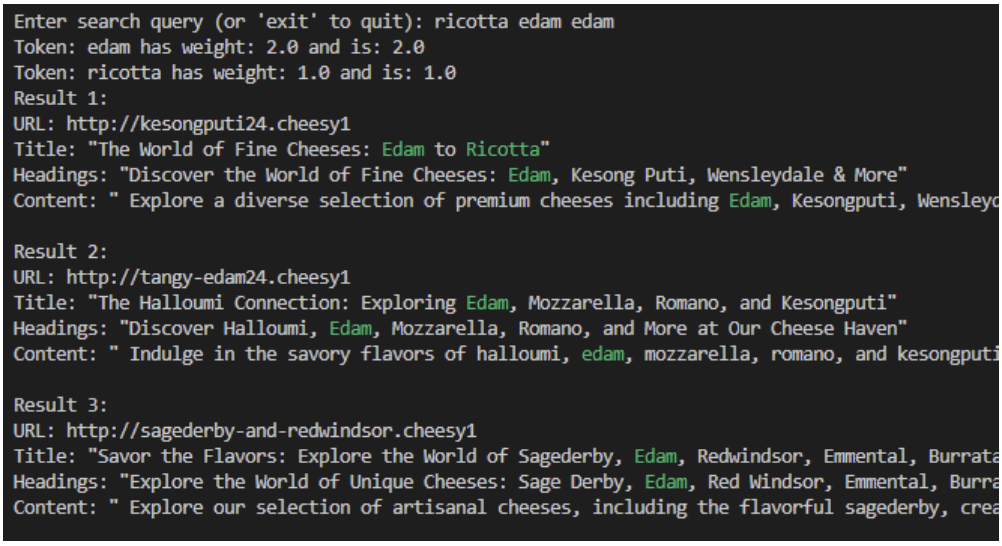
4.2 Reverse Index

Auch hier die Realisierung als eigene Klasse um mögliche Änderungen und Erweiterungen zu erleichtern.
Die Abbildung von Token auf eine Menge von Webseiten ermöglicht es uns komfortabel zu jedem Token eine Liste aller Webseiten zu erhalten, welche dieses enthalten. Diese Funktionalität ist für die Klasse Indexer essentiell und wird genutzt um eine Liste der Suchergebnisse zu erstellen.

Die Kernfunktion searchQuery nutzt zur Vorverarbeitung der Suchbegriff wie Klasse Parser, sodass diese Verarbeitung einheitlich mit den Lemmas im Reverse Index ist.
Anschließend wird über den Reverse Index eine sortierte Menge (doppelt Elimination) der Webseiten erstellt, welche die Suchergebnisse sind. Über einen Parameter kann der searchQuery Funktion mitgeteilt werden, welche Rankingmethode genutzt werden soll. Die searchQuery Funktion wählt die gewünschte Rankingmethode und übergibt dieser die Searchtoken und die Menge der Webseiten, welche Suchergebnisse sind. Die jeweilige Rankingmethode liefert die Suchergebnisse sortiert, absteigend nach dem jeweiligen Score, zurück. Diese Struktur ermöglicht eine Übersicht über alle angebotenen Rankingmethoden und ermöglich es leicht neue einzubinden.

Für die Rankingmethode Kosinusähnlichkeit kann eine Gewichtung der Token übergeben werden (Zusatzaufgabe 3.1). Die Gewichtung findet über die Vielfachkeit der Suchtoken statt, sucht man nach "complexity ricotta complexity", wird dem Suchtoken "complexity"die Gewichtung 2 und "ricotta"die Gewichtung 1 zugewiesen. Dies kann die Reihenfolge der Suchergebnisse verändern. Beispiel:





Ein Nutzer hat somit die Möglichkeit die Gewichtung zu beeinflussen.

Die Rankingmethoden TF-IDF, Kosinusähnlichkeit und die Kombination aus Kosinusähnlichkeit und Pagerank sind alle in der Indexer Klasse enthalten. Da der PageRank Algorithmus weitere Datenstrukturen benötigt wurde er als eigene Klasse realisiert.

Für die Berechnung der Kosinusähnlichkeit ist die in Zusatzaufgabe 3.2 vorgeschlagene Optimierung realisiert. Es sind alle Vektoren auf die Länge 1 normalisiert, auch der Suchvektor wird auf diese Länge normalisiert. Eine Prüfung der Normalisierungfunktion wurde in CosineTests.java hinzugefügt. Somit wird aus:

$$\frac{a \cdot b}{||a|| \cdot ||b||} = \frac{a \cdot b}{1 \cdot 1} = a \cdot b$$

Für die Kombination aus Kosinusähnlichkeit und Pagerank (Aufgabe 5) werden für beide Methoden erst die jeweiligen Scores der Suchergebnisse errechnet. Da der Durchschnittliche Pagerank Score $\frac{1}{N}$, N = Anzahl der Seiten im Netz ist, hat die Anzahl der Seiten im Netz großen Einfluss auf diesen Wert. Aus diesem Grund normalisieren wir beide Scores. Es wird der Durchschnitt über die Scores der Kosinusähnlichkeit errechnet, anschließend werden die Scores in einen Faktor des Durchschnitts umgerechnet. Das gleiche analog für die Pagerank Scores. So wird verhindert, dass die Anzahl der Seiten im Netz, Einfluss auf Gewichtung zwischen Kosinusähnlichkeit und Pagerank nimmt. Ohen diese Normalisierung hätte der Pagerank Score in großen Netzen weniger Bedeutung. Eine Seite mit einem überdurchschnittlichen Pagerank hat hier auch einen Wert über 1 (vorausgesetzt alle anderen Suchergebnisse liegen nicht noch stärker über dem Schnitt).
Anschließend werden beide Scores mit einem Faktor zu einem kombinierten Gesamtscore addiert.
Der Kosinus Score fließt zu 75% ein, der Pagerank Score zu 25%, somit wird der inhaltlichen Übereinstimmung eine größere Bedeutung beigemessen.

Zusätzlich kann die Klasse eine Liste der Token, welche am häufigsten vorkommen, ausgegeben werden mit der Funktion printMostCommonToken. Die Funktion wurde hinzugefügt, durch ein falsches Verständnis von Stoppwörtern. Stoppwörter kommen laut Aufgabentext sehr häufig vor, allerdings ist das häufige vorkommen nicht das einzige Kriterium.

4.3 Klasse Pagerank

Um den Pagerank zu errechnen ist unter anderem eine Datenstruktur oder Funktion notwendig, welches für eine gegebene Webseite, alle Webseiten liefert, welche auf diese verlinken. Aus diesem Grund ist diese Rankingmethode als Klasse implementiert.
Da diese Funktion im Rahmen des Algorithmus häufig genutzt wird, erstellt die Klasse bei der instanziierung eine Datenstruktur, welche diese Information speichert und bereit stellt, sodass die eingehenden Links nicht jedes mal neu ermittelt werden müssen. Diese Datenstruktur speichert die eingehenden Links als Menge um eine Doppeleliminination zu gewährleisten.
Die Pagerank Werte werden in einer Datenstruktur gespeichert, welche die URL der Website mit dem Pagerank Score verknüpft. Diese wird auch mit jeder Iteration neu berechnet und die alte verworfen. Die Summe aller Beträge der Differenzen zwischen neuen und alten Werten wird erfasst und es wird itereriert bis ein Schwellenwert unterschritten ist.

4.4 Record Searchresult

Bündelt die URL, das Page Object und den errechnen Score in ein Objekt um komfortablen Zugriff zu ermöglichen.

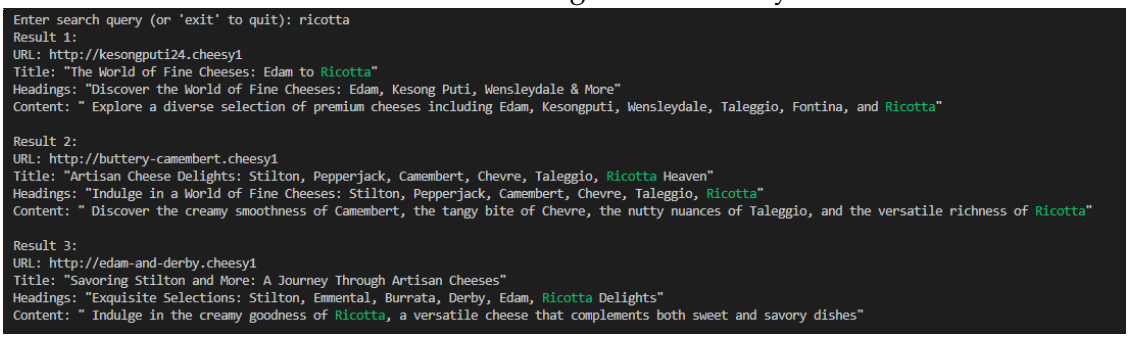
5 Klasse AnguillaSearch (Main Klasse)

Die Mainklasse ist vor allem als Schnittstelle zwischen dem Nutzer und den Funktionen, welcher unser Indexer bietet konzipiert. Sie initialisiert eine Indexer Instanz mit dem gewünschten Netz. Diese Indexer Instanz nutzt wie schon beschrieben den Crawler und die anderen Klassen. Ist das crawlen abgeschlossen, wartet das Programm auf eine Usereingabe. Ist diese kein exit, so leitet es diese Anfrage an die Indexer Instanz weiter, welche die Suchergebnisse verarbeitet. Das Aufbereiten für den Endnutzer findet Mainklasse statt. Die Main Klasse orientiert in der Bedienung am Beispiel aus der Aufgabenstellung.

Zusätzlich zum Titel wird der erste Satz aus dem Inhaltsteil der Webseite ausgegeben, welcher eines der Suchtokens enthält. Dies funktioniert in den meisten Fällen einwandfrei, allerdings, nicht in 100% der Fälle. Die Funktion verwendet die lemmatisierten Suchtokens, diese stimmen nicht immer mit unlemmatisierten Versionen überein. Um die Funktionalität zu verbessern, müsste ein Abgleich mit den Lemmas, welche auf der

Webseite enthalten sind stattfinden und diese anschließend wieder dem Satz zugeordnet werden. Durch die Klasse Page liegt zwar eine Liste aller Lemmas vor, allerdings ist eine anschließende Zuordnung zum jeweiligen Satz nicht trivial.

Ebenfalls untersützt das Programm eine farbliche Hervorhebung der Wörter die zum Match geführt haben, allerdings funktioniert auch diese Hervorhebung nicht zu 100%, aufgrund der gleiche Problematik lemmatisiertes Token und unlemmatisierter Inhalt. Hier eine Suche mit farblicher Hervorhebung im Netz cheesy1.



Zusätzlich können über Kommandozeilenparameter einige Aspekte des Programm konfiguriert werden. Unter anderem unterstützt das Programm. Unterstützt wird die Auswahl der Rankingmethode über '-r Ziffer'. 0 wählt TF-IDF, 1 wählt die Kosinusähnlichkeit, 2 die Kombination aus Kosinusähnlichkeit und Pagerank und 3 die Kosinusähnlichkeit mit Gewichten.
Die farbliche Hervorhebung kann durch –color aktiviert werden und funktioniert nur in einer Kommandozeile, welche die ANSI-Farben unterstützt und ist standardmäßig deaktiviert.
Das Netz, welches durchsucht werden soll wird über den letzten Parameter übergeben, unterstützt wird das übergeben eines Dateipfads zu einer JSON-Datei oder das übergeben der SeedURLs getrennt von Leerzeichen, als ein Parameter.
Beispiel, wir wollen Farben und die Kosinusähnlichkeit mit Gewicht als Rankingmethode. Wir übergeben die seedURLs vom Netz cheesy2.

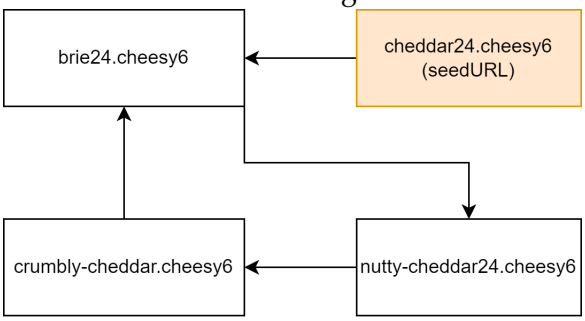
```
java -jar ./target/anguillasearch-1.0.0-SNAPSHOT.jar -color -r 3
"http://shropshireblue24.cheesy2 http://burrata.cheesy2 http://stilton24.cheesy2"
```

6 Aufgabe 4.3

Die Ausgabe ist in Git als Ausgabe 4.3.txt gespeichert.
Hier ein Ausschnitt:

```
Iteration: 0----------------------------
URL: http://brie24.cheesy6                          old PR: 0.250000 new PR: 0.500000
URL: http://cheddar24.cheesy6                       old PR: 0.250000 new PR: 0.000000
URL: http://crumbly-cheddar.cheesy6                 old PR: 0.250000 new PR: 0.250000
URL: http://nutty-cheddar24.cheesy6                 old PR: 0.250000 new PR: 0.250000
Iteration: 1----------------------------
URL: http://brie24.cheesy6                          old PR: 0.500000 new PR: 0.250000
URL: http://cheddar24.cheesy6                       old PR: 0.000000 new PR: 0.000000
URL: http://crumbly-cheddar.cheesy6                 old PR: 0.250000 new PR: 0.250000
URL: http://nutty-cheddar24.cheesy6                 old PR: 0.250000 new PR: 0.500000
Iteration: 2----------------------------
URL: http://brie24.cheesy6                          old PR: 0.250000 new PR: 0.250000
URL: http://cheddar24.cheesy6                       old PR: 0.000000 new PR: 0.000000
URL: http://crumbly-cheddar.cheesy6                 old PR: 0.250000 new PR: 0.500000
URL: http://nutty-cheddar24.cheesy6                 old PR: 0.500000 new PR: 0.250000
Iteration: 3----------------------------
URL: http://brie24.cheesy6                          old PR: 0.250000 new PR: 0.500000
URL: http://cheddar24.cheesy6                       old PR: 0.000000 new PR: 0.000000
URL: http://crumbly-cheddar.cheesy6                 old PR: 0.500000 new PR: 0.250000
URL: http://nutty-cheddar24.cheesy6                 old PR: 0.250000 new PR: 0.250000
Iteration: 4----------------------------
URL: http://brie24.cheesy6                          old PR: 0.500000 new PR: 0.250000
URL: http://cheddar24.cheesy6                       old PR: 0.000000 new PR: 0.000000
URL: http://crumbly-cheddar.cheesy6                 old PR: 0.250000 new PR: 0.250000
URL: http://nutty-cheddar24.cheesy6                 old PR: 0.250000 new PR: 0.500000
Iteration: 5----------------------------
URL: http://brie24.cheesy6                          old PR: 0.250000 new PR: 0.250000
URL: http://cheddar24.cheesy6                       old PR: 0.000000 new PR: 0.000000
URL: http://crumbly-cheddar.cheesy6                 old PR: 0.250000 new PR: 0.500000
URL: http://nutty-cheddar24.cheesy6                 old PR: 0.500000 new PR: 0.250000
```

Wie zu erwarten ist hat die seedURL cheddar24.cheesy6 bereits nach einer Iteration einen Pagerank von 0, da keine andere Webseite auf diese verlinkt. Dies ändert sich auch nicht.
Iteration 1, 4, 7, 11 usw. errechnen die gleichen Pagerank Werte, somit befindet sich das Programm in einer Endlosschleife, da die kumulierten Beträge der Differenzen zwischen neuen und altem Pagerank nicht unter den Schwellenwert sinken können.



Wie auf dem Bild zu sehen ist, handelt fast 1:1 um denselben Netzzyklus wie in Abbil-

dung 3.4 der Aufgabenstellung. Auch an den Werten lässt sich ablesen, dass die 0.25 Pagerank zyklisch von brie24 zu nutty-cheddar24 zu crumbly-cheddar und wieder zu brie24 wandern.

7 Zusatzaufgabe 2

Das Token “caerphilly” ist in jeder Webseite enthalten, somit ist der IDF Wert 0 und folglich auch der TF-IDF Wert.
Es wurde eine Suchanfrage mit dem token “experience” durchgeführt, da dieses Token nicht in jeder Webseite vorkommt.