UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și tehnologia informației
SPECIALIZAREA: Tehnologia informației

Proiect la materia
Regăsirea Informațiilor pe Web
Etapa I

Profesor coordonator:

Ș. l. dr. ing. Alexandru Archip

Student:

Socea Gabriel - grupa 1410B

Iași, 2020

Cuprins

1. Introducere

Această aplicație reprezintă prima etapa a proiectului de la materia “Regasirea Informațiilor pe Web“. Această etapă presupune crearea indecșilor direct și invers care sunt stocați folosind baza de date nerelațională MongoDB, precum și procesarea cuvintelor înainte de a fi stocate și procesul de căutare vectorială, care asociază un scor fiecărui document care se califică pentru afișare și în funcție de acel scor se vor afișa documentele.

Implementarea a fost realizată in limbajul de programare de nivel înalt Java. Codul a fost scris in mediul de dezvoltare Eclipse. Pentru lucru cu baza de date MongoDB am folosit biblioteca mongo_java_driver_3.0.4.jar și bineînțeles serverul Mongo.

1. Preprocesarea cuvintelor

Pentru fiecare cuvânt găsit dintr-un fișier se testează mai întâi dacă acesta este excepție folosind o listă de cuvinte - excepții care sunt stocate într-un fișier text. Excepțiile sunt cuvintele care nu trebuie modificare, cel mai adesea acestea fiind nume de persoane, nume de țari, orașe, regiuni, nume de locuri, clădiri, monumente, firme, acronime etc. Daca cuvântul respectiv este excepție atunci el este stocat în baza de date fără a efectua modificări asupra sa.

Dacă nu este excepție atunci se testează în continuare daca acesta este “stop word“ folosind la fel ca la excepții o lista stocata într-un fișier text. Daca acest este excepție atunci se ignoră, nefiind băgat in baza de date.

Daca nu este nici “stop word“ atunci înseamnă că trebuie adus la forma canonică, adică la forma sa simplă. Acest lucru se realizează cu ajutorul algoritmului lui Porter.

2. Forma canonica a cuvintelor – Algoritmul lui Porter

Pentru a aduce cuvântul la forma canonică am folosit algoritmul lui Porter. Acest algoritm are 5 pași. Primul pas are și el trei mici pași in care se adaugă, se scot, sau se înlocuiesc una sau mai multe litere de la sfârșitul cuvântului. La pasul al doilea, daca conține un sufix anume din acea lista, atunci acesta se înlocuiește cu sufixul indicat. La pasul al treilea si al patrulea se înlocuiesc unele sufixe cu altele indicate, sau chiar se șterg acele sufixe de la sfârșitul cuvintelor. Pasul cinci este împărțit și el in doi mici subpași în care se scoate o litera de la sfârșitul cuvântului.

Trebuie menționat că la toți acești pași mai sunt adăugate câteva condiții adiționale (ex: m>0 sau m>1, *v*, *S, *d, *o etc.).

Am ales acest algoritm de steeming și nu altul deoarece este mai popular, deci se găsesc mai multe informații despre acesta și este si mai recent fața de alți algoritmi de genul acesta.

3. Indexul direct stocat in baza de date MongoDB

In constructorul clasei care realizează indexul direct se creează o conexiune cu baza de date Mongo, se creează o instanța a bazei de date folosite din mulțimea de baze de date, se creează colecția “Index Direct“ și se inițializează toate variabilele folosite in clasă.

Apoi avem o funcție care verifica daca String-ul dat ca parametru este excepție și încă o funcție care verifică dacă un cuvânt este “stop word“, dar si o funcție care creează un BufferedReader pe baza unui fișier din colecția de fișiere de intrare.

Mai departe avem o funcție care creează un HashTable care conține cuvintele preprocesate si numărul de apariții ale lor dintr-un fișier. Mai avem o funcție care parcurge lista de fișiere de intrare din coada de fișiere si le procesează, pentru fiecare apelând funcția de creare a HashTable-ului si apoi inserând rezultatul in baza de date.

Funcția cea mai importanta din aceasta clasa, care este si apelata in clasa de testare este funcția care parcurge structura de directoare iterativ si apelează funcția de mai devreme, dar unde se creează si documentul mapper direct.

Pentru a stoca documentele in baza de date avem o funcție care face acest lucru si care este apelata in funcțiile de mai sus.

Trebuie menționat si faptul ca in indexul direct sunt stocate pentru fiecare cuvânt din fiecare document si tf (term frequency), variabila care ajuta la realizarea căutării vectoriale si a relevantelor determinate in cadrul acesteia.

4. Indexul invers stocat in baza de date MongoDB

In constructorul clasei care realizează indexul invers se fac toate setarile pentru folosirea bazei de date si se inițializează toate obiectele si variabilele primitive folosite in acesta clasa. Se folosesc doua HashMap-uri pentru realizarea acestui index, unul pentru a creea indexul invers la nivel de fișiere dintr-un folder si celălalt la crearea indexului invers total adică pentru documentul care conține toate cuvintele din toate fișierele.

O funcție destul de importanta in aceasta clasa este funcția care asignează valori la cele doua HashMap-uri interne pe baza indexului direct creat. Încă o funcție importanta ar fi ProcessingFilesReverse() care procesează lista de documente corespunzătoare indexului direct care apelează funcția de mai devreme, inserează valori in baza de date si totodată creează fi documentul mapper invers.

Apoi avem o funcție care scrie un document in colecția de documente a indexului invers, funcție care este apelata in ProcessingFilesReverse().

In cadrul acestui index la documentul de index total se inserează pentru fiecare cuvânt care apare si valoarea idf (Inverse Document Frequency) valoare ajutătoare la căutarea vectoriala.

5. Cautarea vectoriala in baza de date MongoDB

In constructorul acestei clase, la fel ca la celelalte clase, se instanțiază obiectele necesare in lucrul cu baza de date.

Avem o funcție care procesează lista de cuvinte cheie din căutare, adică se adaugă într-o lista cuvintele cheie si in alta lista operațiile dintre cuvinte (OR, AND,NOT). Dar si aici la fel ca la indexarea directa pentru fiecare cuvânt exista partea de preprocesare care verifica pe rând daca cuvântul este excepție, sau stop word si daca nu se aduce cuvântul la forma canonica cu ajutorul algoritmului lui Porter.

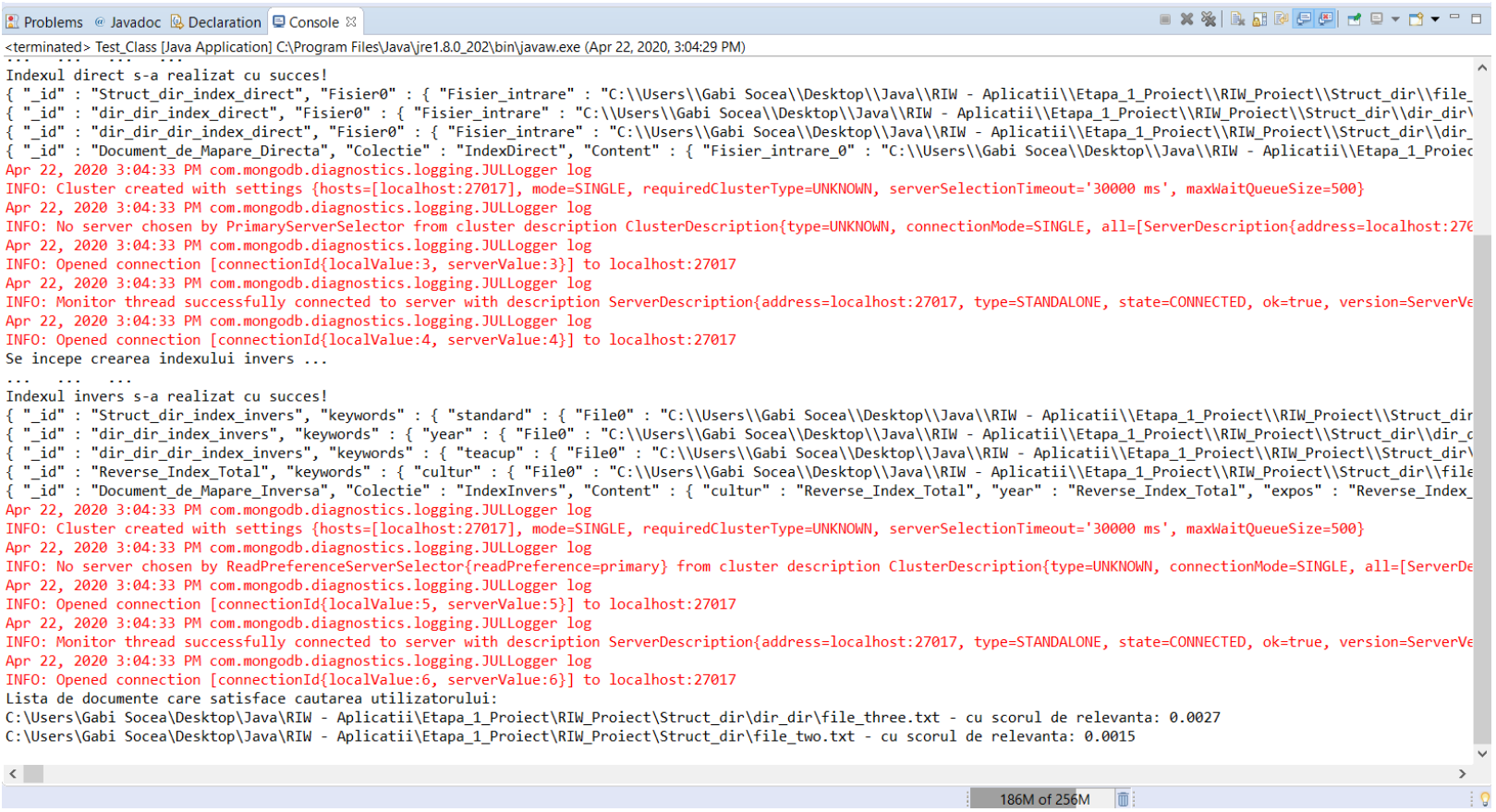
Apoi se aduc intr-un HashMap<String, ArrayList<String>> pentru fiecare cuvânt, calea tuturor fișierelor in care acesta apare. Aceasta este singura funcție care folosește baza de date.

Funcția care realizează căutarea vectoriala VectorialSearch() in care se calculează similaritatea cosinus pentru fiecare doua documente si apoi sortează documentele in funcție de aceasta valoare. Aceasta funcție apelează alte funcții necesare procesului de căutare, funcții precum sortDocuments() care sortează documentele in funcție de relevanta, DocumentNorm() care calculează norma euclidiana a unui document, SearchQueryNorm() care calculează norma euclidiana a cuvintelor din cautare si funcția MultiplyVectors() care calculează de fapt numărătorul din formula similarității cosinus.

In afara de aceste funcții mai avem încă șase funcții care realizează aplicarea operatorilor, dintre care doua sunt pentru AND, doua sunt pentru OR si celelalte doua sunt pentru NOT. Sunt cate doua funcții pentru fiecare operație pentru ca una din ele realizează operația intre mulțimile de documente ale celor doua cuvinte primite ca parametru si cealaltă realizează operația intre lista de fișiere a unui cuvânt dat ca parametru si o alta lista de documente tot la fel primita ca parametru.

6. Testarea aplicație si rezultate obținute

Pentru testarea aplicației s-a creat o clasa noua care conține metoda Main si s-au apelat modulele implementate. Rezultatele căutării vectoriale se afișează la consola fiind sortate in funcție de rank-ul obținut, totodată fiind afișat si rank-ul.


Problems
Javadoc
Declaration
Console
<terminated> Test_Class [Java Application] C:\Program Files\Java\jre1.8.0_202\bin\javaw.exe (Apr 22, 2020, 3:04:29 PM)
... ...
Indexul direct s-a realizat cu succes!
{ "_id" : "Struct_dir_index_direct", "Fisier0" : { "Fisier_intrare" : "C:\\Users\\Gabi Socea\\Desktop\\Java\\RIW - Aplicatii\\Etapa_1_Proiect\\RIW_Proiect\\Struct_dir\\file_
{ "_id" : "dir_dir_index_direct", "Fisier0" : { "Fisier_intrare" : "C:\\Users\\Gabi Socea\\Desktop\\Java\\RIW - Aplicatii\\Etapa_1_Proiect\\RIW_Proiect\\Struct_dir\\dir_dir\
{ "_id" : "dir_dir_dir_index_direct", "Fisier0" : { "Fisier_intrare" : "C:\\Users\\Gabi Socea\\Desktop\\Java\\RIW - Aplicatii\\Etapa_1_Proiect\\RIW_Proiect\\Struct_dir\\dir_
{ "_id" : "Document_de_Mapare_Directa", "Colectie" : "IndexDirect", "Content" : { "Fisier_intrare_0" : "C:\\Users\\Gabi Socea\\Desktop\\Java\\RIW - Aplicatii\\Etapa_1_Proiec
Apr 22, 2020 3:04:33 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType=UNKNOWN, serverSelectionTimeout='30000 ms', maxWaitQueueSize=500}
Apr 22, 2020 3:04:33 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: No server chosen by PrimaryServerSelector from cluster description ClusterDescription{type=UNKNOWN, connectionMode=SINGLE, all=[ServerDescription{address=localhost:276
Apr 22, 2020 3:04:33 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Opened connection [connectionId{localValue:3, serverValue:3}] to localhost:27017
Apr 22, 2020 3:04:33 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Monitor thread successfully connected to server with description ServerDescription{address=localhost:27017, type=STANDALONE, state=CONNECTED, ok=true, version=ServerVe
Apr 22, 2020 3:04:33 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Opened connection [connectionId{localValue:4, serverValue:4}] to localhost:27017
Se incepe crearea indexului invers ...
... ...
Indexul invers s-a realizat cu succes!
{ "_id" : "Struct_dir_index_invers", "keywords" : { "standard" : { "File0" : "C:\\Users\\Gabi Socea\\Desktop\\Java\\RIW - Aplicatii\\Etapa_1_Proiect\\RIW_Proiect\\Struct_dir
{ "_id" : "dir_dir_index_invers", "keywords" : { "year" : { "File0" : "C:\\Users\\Gabi Socea\\Desktop\\Java\\RIW - Aplicatii\\Etapa_1_Proiect\\RIW_Proiect\\Struct_dir\\dir_d
{ "_id" : "dir_dir_dir_index_invers", "keywords" : { "teacup" : { "File0" : "C:\\Users\\Gabi Socea\\Desktop\\Java\\RIW - Aplicatii\\Etapa_1_Proiect\\RIW_Proiect\\Struct_dir\
{ "_id" : "Reverse_Index_Total", "keywords" : { "cultur" : { "File0" : "C:\\Users\\Gabi Socea\\Desktop\\Java\\RIW - Aplicatii\\Etapa_1_Proiect\\RIW_Proiect\\Struct_dir\\file
{ "_id" : "Document_de_Mapare_Inversa", "Colectie" : "IndexInvers", "Content" : { "cultur" : "Reverse_Index_Total", "year" : "Reverse_Index_Total", "expos" : "Reverse_Index_
Apr 22, 2020 3:04:33 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType=UNKNOWN, serverSelectionTimeout='30000 ms', maxWaitQueueSize=500}
Apr 22, 2020 3:04:33 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: No server chosen by ReadPreferenceServerSelector{readPreference=primary} from cluster description ClusterDescription{type=UNKNOWN, connectionMode=SINGLE, all=[ServerDe
Apr 22, 2020 3:04:33 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Opened connection [connectionId{localValue:5, serverValue:5}] to localhost:27017
Apr 22, 2020 3:04:33 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Monitor thread successfully connected to server with description ServerDescription{address=localhost:27017, type=STANDALONE, state=CONNECTED, ok=true, version=ServerVe
Apr 22, 2020 3:04:33 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Opened connection [connectionId{localValue:6, serverValue:6}] to localhost:27017
Lista de documente care satisface cautarea utilizatorului:
C:\Users\Gabi Socea\Desktop\Java\RIW - Aplicatii\Etapa_1_Proiect\RIW_Proiect\Struct_dir\dir_dir\file_three.txt - cu scorul de relevanta: 0.0027
C:\Users\Gabi Socea\Desktop\Java\RIW - Aplicatii\Etapa_1_Proiect\RIW_Proiect\Struct_dir\file_two.txt - cu scorul de relevanta: 0.0015
186M of 256M