

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

Indexare și căutare

Profesor îndrumător
Ș.l. dr. ing. Alexandru Archip

Student
Poeană Ionuț
Grupa 1410B

Iași, 2020

Cuprins

Enunțul proiectului.....	1
Capitolul 1. Prezentarea generală a aplicației propuse ca soluție.....	2
Capitolul 2. Modulele aplicației.....	3
2.1. Index direct + „tf”.....	3
2.2. Algoritmi de stemming.....	4
2.2.1. Algoritmul de stemming al lui Lovins.....	4
2.2.2. Algoritmul de stemming al lui Porter.....	4
2.3. Index indirect + „idf”.....	5
2.4. Creare vectorilor asociați.....	5
2.5. Căutarea vectorială.....	6
Capitolul 3. Viteza de procesare.....	8
3.1. Viteză de creare index direct + tf.....	8
3.2. Viteză creare index indirect + idf.....	8
3.3. Viteză creare vectori asociați.....	8
3.4. Viteză încărcare vectori asociați în memorie.....	8
3.5. Viteza de căutare vectorială.....	9

Enunțul proiectului

Prima componentă de proiect a disciplinei Regăsirea Informațiilor pe WEB vă propune să realizați un set de modificări asupra aplicațiilor dezvoltate pe parcursul laboratoarelor 1-4 astfel încât:

1. să obțineți o procesare adecvată a cuvintelor determinate în cadrul unui text;
2. să studiați și alte tipuri de baze de date și mecanisme de stocare de date (precum MongoDB);
3. să studiați alte metode de realizare a operațiilor de căutare.

Astfel, pentru prima cerință de mai sus, metodele de construire a indecșilor direcți și inverși trebuie să implementeze mecanisme prin intermediul cărora un cuvânt de dicționar să fie adus la așa numita *formă canonică*. Indecșii astfel determinați, vor fi apoi stocați prin intermediul unor baze de date ne-relaționale, precum MongoDB.

După cum a fost amintit, căutarea booleană nu include mecanisme de determinare a unui eventual scor de relevanță pentru rezultatele identificate. A treia cerință vă propune să studiați impactul *distanței de tip cosinus* asupra rezultatelor obținute de un motor de căutare.

1.1. Evaluare proprie după baremul afișat

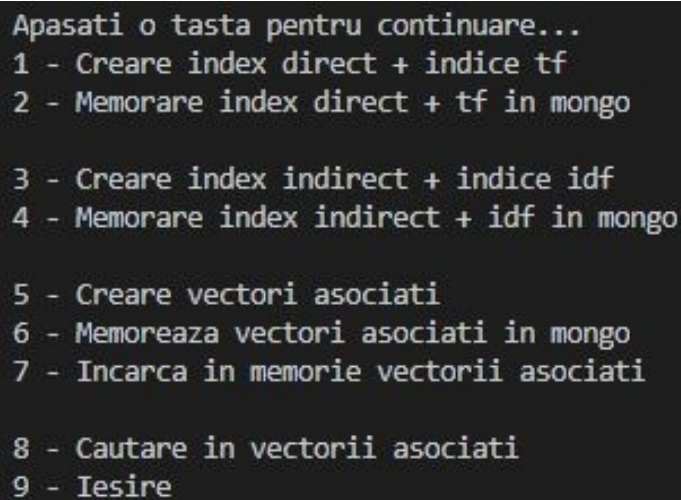
1. Parcurgerea directorului de fișiere de indexat într-o manieră iterativă **0.5 puncte**;
2. Preprocesarea, aplicarea unui algoritm de stemming pentru determinarea formelor canonice ale cuvintelor determinate **2 puncte**, se aplică algoritmul de stemming al lui Porter, acesta constă în aplicarea secvențială a unui set de reguli asupra cuvintelor, pentru eliminarea sufixelor și transformarea terminației acestora, pe baza mărimii acestora (valoarea parametrului m). Structura de indexare directă se memorează sub forma unei colecții în mongoDB cu numele **directIndex**;
3. Indexarea indirectă **3.5 puncte**, la fel ca la indexarea directă, indexarea indirectă se realizează local, pentru fiecare document din colecția **directIndex** în parte. Se creează și o structură de indexare indirectă global pentru a se putea calcula **inverse document frequency-ul**. Structura de indexare indirectă se memorează sub forma unei colecții în mongoDB cu numele **indirectIndex**;
4. Căutarea în indecși determinați preîncărcați **2.5 puncte**, se realizează căutarea vectorială cu restrângerea colecției de lucru. Similaritatea de tip cosinus se calculează doar între documentul introdus ca interogare și documentele care conțin cel puțin un cuvânt din cele prezente în interogare. De asemenea, similaritatea de tip cosinus este ponderată la numărul de cuvinte din interogare care se găsesc în fișiere.
5. Inserarea într-o bază de date nerelatională a datelor necesare, s-a folosit MongoDB, **1.5 puncte**.
6. Paralelizari eficiente a volumului de lucru **3 puncte**. S-a realizat paralelizarea volumului de lucru la etapele de creare a indexului direct și a vectorilor asociați documentelor. Astfel timpul de procesare s-a redus la mai puțin de jumătate în unele cazuri:
 1. calcularea indexului direct: de la 1.091 secunde la 0.013 secunde;
 2. calcularea vectorilor asociați: de la 0.56 secunde la 0.354 secunde;

Notă evaluare student: 14.

Capitolul 2. Prezentarea generală a aplicației propuse ca soluție

Aplicația propusă a fost scrisă în limbajul Java, utilizând mediul de dezvoltare Visual Studio Code. În scrierea soluției s-a utilizat o librărie externă pentru a facilita lucrul cu baza de date de tip mongoDB.

Interfața cu utilizatorul este simplă, aceasta se folosește de funcționalitățile liniei de comandă și este implementată cu ajutorul structurii de decizie *switch-case*.



```
Apasati o tasta pentru continuare...
1 - Creare index direct + indice tf
2 - Memorare index direct + tf in mongo

3 - Creare index indirect + indice idf
4 - Memorare index indirect + idf in mongo

5 - Creare vectori asociati
6 - Memoreaza vectori asociati in mongo
7 - Incarca in memorie vectorii asociati

8 - Cautare in vectorii asociati
9 - Iesire
```

Figura 2.1: Meniul utilizator

Capitolul 3. Modulele aplicației

3.1. Index direct + „tf”

Acest modul creează index-ul direct al tuturor documentelor *HTML* sau text găsite în directoarele și subdirectoarele directorului sursă. Pentru parcurgerea directoarelor, este folosită o coadă, deoarece se dorește o parcurgere secvențială și nu recursivă (se dorește eliminarea posibilității ca la parcurgerea structurii de directoare să apară excepții de tip „Stack Overflow”).

Pentru fiecare intrare din structura de directoare, se verifică dacă aceasta este director sau fișier:

- dacă este director, acesta este adăugat în coadă, conținutul acestuia va fi procesat mai târziu;
- dacă este fișier, acesta este procesat. Procesarea constă în verificarea tipului de text conținut, aplicația poate procesa fișierele de tip *TXT*.

Textul conținut în fișiere este procesat caracter cu caracter pentru a se atinge o performanță cât mai ridicată în prelucrarea acestora. Se extrag cuvintele din text, care mai apoi sunt testate împotriva unor seturi de cuvinte *excepții* și *stopwords*.

1. dacă cuvântul curent este o *excepție* (cuvânt ce prezintă interes, dar nu se găsește în dicționar, trebuie tratat ca atare) îl adăugăm la indexul direct;
2. dacă cuvântul curent este un *stopword* (cuvânt ce nu prezintă interes pentru relevanța documentelor rezultate din căutare, este ignorat) nu va apărea în index-ul direct nici în viitoarele derivări ale acestuia (index-ul indirect);
3. dacă cuvântul curent este un cuvânt de dicționar acesta se trece printr-un algoritm de **stemming**, în cazul acesta algoritmul lui Porter, pentru a se elimina terminațiile a fi adus la o formă canonică.

Cuvintele care trec de procesare sunt memorate într-un fișier de mapare numit de tip index direct de forma `<nume_fisier:<cuvant1,nr_aparitii>, <cuvant2,nr_aparitii>, ...>`.

În acest modul se creează și fișierul *term_frequency* corespunzător, datorită ușurinței cu care acesta se poate calcula la acest pas, avem acces la fiecare cuvânt și la numărul de apariții ale acestuia în cadrul fișierului. „tf”-ul reprezintă frecvența de apariție a cuvântului k în conținutul documentului.

P1 – frecvența de apariție a cuvântului k în cadrul documentului d (în eng.: **term frequency**)

$$tf(k, d) = \frac{count(k)}{|d|}$$

Figura 3.1: Formula de calcul a „term frequency”

3.2. Algoritmi de stemming

Procese prin care un cuvânt este adus la forma sa de bază poartă numele de **stemming** și **lemantizare**. Cele două modele de procesare urmăresc același scop: reducerea dimensiunilor indecșilor. Pentru cuvintele ce aparțin de dicționarul unei limbi nu are sens stocarea tuturor formelor sub care aceste cuvinte pot fi derivate/conjugate (deoarece aceste transformări nu aduc efectiv informații noi procesului de indexare).

Ambele metode, **stemming** și **lemantizare**, generează forma canonică a unui cuvânt însă principala diferență este că **stem**-urile, uitățile de bază din algoritmi de stemming nu sunt cuvinte de dicționar, în timp ce **lemma**, unitățile de bază din algoritmi de lemantizare sunt cuvinte de dicționar.

Modelele de analiză bazate pe **lemantizare** sunt axate pe analiză morfologică și teoretic ar trebui să ofere rezultate cu un grad ridicat de precizie. Totuși, procesele sunt lente, considerabil mai greoi de implementat și costisitoare computațional.

Procesele de tip **stemming** se bazează pe trunchierea cuvintelor (de exemplu, în limba engleză pluralul cuvintelor se obține în majoritatea cazurilor prin adăugarea caracterului „s”). Aceste tehnici sunt considerabil mai simple, cu un timp de răspuns mult mai redus, dar nu oferă aceeași precizie. Unul dintre cei mai renumiți algoritmi este algoritmul lui Porter[1].

3.2.1. Algoritmul de stemming al lui Lovins

Algoritmul de stemming al lui Lovins (1968) a fost primul algoritm de acest tip publicat. Este mult mai mare decât algoritmul lui Porter din cauza listei foarte extinse de sufixe. Are în componența acestuia 2 pași majori în timp ce algoritmul lui Porter are 8 pași majori. Comparabil cu algoritmul lui Porter, acesta este mai rapid[2].

Algoritmul prezintă o listă de 294 de sufixe condiționate de 29 de reguli pentru eliminarea acestora și 35 de reguli de transformare. Fiecare sufix este asociat cu o condiție. În primul pas este eliminat cel mai lung sufix care satisface condiția asociată. În cel de-al doilea pas se aplică cele 35 de reguli pentru a transforma sfârșitul cuvântului indiferent dacă acesta a suferit sau nu modificări în prima etapă.

Spre exemplu: cuvântul **nationally** are sufixul **ationally**, cu condiția asociată B: „lungimea minimă a stem-ului = 3”. Din moment ce prin eliminarea sufixului **ationally** lungimea stem-ului rămas ar fi de 1, condiția B împiedică eliminarea acestuia. În același timp cuvântul conține sufixul **ionally**, ce are asociată condiția A „nicio restricție pentru lungimea stem-ului”. Astfel se elimină sufixul **ionally** și se trece la cel de-al doilea pas din algoritm.

Regulile de transformare gestionează caracteristici precum dublarea literelor (sitting - sitt – sit), forme de plural neregulate (matrix și matrices) și neregularități morfologice ale limbii engleze cauzate de conjugarea verbelor latine la a 2-a formă (assume – assumption, commit – commission etc). Deși sunt descrise ca aplicate secvențial, regulile pot fi împărțite în 2 etape, regula 1 se poate aplica în prima etapă, regulile de la 0 până la 35, exceptând 1, se pot aplica în etapa a 2-a.

3.2.2. Algoritmul de stemming al lui Porter

Algoritmul de stemming al lui Porter este un proces de eliminare a sufixelor din limba engleză. Acesta a fost construit pentru îmbunătățirea performanțelor sistemelor de regăsire a informațiilor, sub presupunerea că nu dispunem de un dicționar cu formele de bază ale cuvintelor[3].

Scopul algoritmului este de aduce împreună documente care conțin aceeași formă canonică a cuvintelor. Spre exemplu dacă avem 2 documente, primul document conține cuvântul **connection** și cel de-al doilea cuvântul **connections** este extrem de probabil ca utilizatorul în căutarea unor informații despre termenul **connection** să găsească date relevante și în cel de-al doilea document, nu doar în primul. Astfel, prin aducerea termenilor la forma canonică, sistemul de regăsire a informațiilor poate furniza date mai complete la căutarea acestora.

Rata de succes pentru eliminarea sufixului va fi semnificativ mai mică de 100% indiferent de modul de evaluare al algoritmului. De exemplu, dacă **sand** și **sander** sunt evaluate

la fel, atunci cel mai probabil **wand** și **wander** vor fi evaluate la fel. Eroarea este dată aici de faptul că terminația **-er** a cuvântului **wander** este tratată ca sufix, în timp ce aceasta este parte a rădăcinii cuvântului. În egală măsură, un sufix poate modifica complet sensul unui cuvânt, caz în care îndepărtarea acestuia nu este de folos.

Adăugarea mai multor reguli pentru a face o prelucrare mai precisă a cuvintelor dintr-o anumită parte a vocabularului poate cauza performanțe reduse și apariția erorilor în altă parte a vocabularului.

Algoritmul se bazează pe dimensiunea **m** a cuvântului, **m** reprezintă numărul de grupuri, consoane și vocale alternate, o consoană între 2 vocale sau o vocală între 2 consoane, în cazul în care avem mai multe consoane sau mai multe vocale alăturate, acestea se consideră ca una singură, spre exemplu:

- **m=0** exemplu: tr, ee, tree, y, by;
- **m=1** exemplu: trouble, oats, trees, ivy;
- **m=2** exemplu: troubles, private, oaten, orrery.

Regulile de eliminare a sufixelor se bazează pe valoarea parametrului **m**. Algoritmul are grijă să nu elimine un sufix atunci când rădăcina cuvântului este prea scurtă (parametrul **m** este prea mic). Această abordare nu are o bază lingvistică. Prin încercări s-a observat că se poate folosi măsura unui cuvânt (**m**) pentru a vedea dacă este bine sau nu să se elimine un sufix.

3.3. Index indirect + „idf”

În această etapă este creat index-ul indirect cu ajutorul colecției de documente **directIndex** stocat în mongo. Pentru fiecare cuvânt din documente se realizează maparea inversă: **<cuvant: <nume_fisier1,nume_fisier2>, ...>**.

Documentele de tip **indirectIndex** conțin în structura acestora cuvântul și un vector de stringuri ce reprezintă fișierele în care se găsește cuvântul.

Tot în această etapă este calculat inverse document frequency, **idf-ul**, datorită ușurinței cu care avem acces la datele necesare pentru a aplica formula de calcul. Numărul de documente în care apare cuvântul și numărul total de documente.

P2 – frecvența inversă de apariție a cuvântului *k* în cadrul mulțimii de documente *D* (în eng.: **inverse document frequency)**

$$idf(k) = \log \frac{|D|}{1 + |\{d : k \in d\}|}$$

Figura 3.2: Formula de calcul a „inverse document frequency”

3.4. Creare vectorilor asociați

În cadrul algoritmului de căutare vectorială, atât documentele cât și interogarea trebuie reprezentate sub forma unor vectori ce conțin elemente de tipul **<cuvant, tf*idf>**.

Pentru calcularea vectorilor asociați se folosesc documentele din colecția **directIndex** și **idf**. Se preia fiecare intrare din fișier și se citește fișierul de tip index indirect corespunzător intrării respective. Pentru fiecare document din colecția **directIndex** se preia numele fișierului și valorile corespunzătoare **tf**-ului și **idf**-ului pentru fiecare cuvânt. Produsul **tf*idf** este memorat într-o structură de tip **<HashMap<String,Double>>**, cheia acestei perechi este reprezentată de cuvânt.

Fiecare vector asociat unui document este memorat într-o structură de tipul **HashMap<String, HashMap<String, Double>>** , cheia este reprezentată de numele fișierului, care mai apoi este inserată în mongoDB.

3.5. Căutarea vectorială

Pentru a putea efectua o căutare vectorială este necesar să încărcăm în memorie într-o structură de tipul **HashMap<String, HashMap<String, Double>>** din colecția care memorează vectorii asociați, **associatedVectors**. Încărcarea în memorie se realizează doar la prima căutarea, pentru căutările viitoare din cadrul aceleiași rulări a programului vectorii rămân încărcăți.

Pseudocodul algoritmului de căutare vectorială este prezentat în Figura 4.1.

Algoritm 1 cautareVectoriala(D, q)

- 1: transforma fiecare document $d_j \in D$ în $\vec{d}_j = \{key : tf(key, d) \cdot idf(key)\}$
 - 2: transforma interogarea $\vec{q} = \{key : tf(key, d) \cdot idf(key)\}$
 - 3: **for all** $\vec{d}_j \in D$ **do**
 - 4: calculează $s_j = similaritateCosinus(\vec{d}_j, \vec{q})$
 - 5: **end for**
 - 6: sortează documentele descrescător din punct de vedere al scorului anterior s_j
 - 7: **return** setul relevant de documente
-

Figura 3.3: Pseudocod algoritm de căutare vectorială

Pentru căutare se parsează interogarea dată de utilizator și se calculează vectorul asociat al acesteia. Interogarea este tratată ca un document. Apoi se calculează similaritatea cosinus dintre vectorul interogării și vectorul fiecărui fișier din structura **fileAssociatedVectors** care conține măcar un cuvânt în comun cu interogarea.

$$\cos(\angle \vec{d1}, \vec{d2}) = \frac{\vec{d1} \cdot \vec{d2}}{\|\vec{d1}\| \|\vec{d2}\|}$$

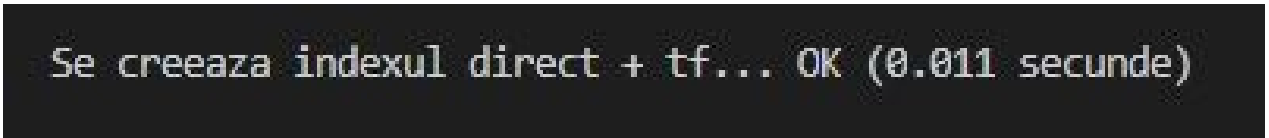
Figura 3.4: Similaritatea cosinus

În final se sortează descrescător perechile <fișier, similaritate_cosinus> rezultate în funcție de valoarea dată de formula de calcul ponderate cu numărul de cuvinte din interogare care s-au găsit în fișiere.

Capitolul 4. Viteza de procesare

Exemplele prezentate în cadrul proiectului au la bază procesarea fișierelor de test din cadrul temei la disciplina Algoritmi Paraleli și Distribuți, modelul mapReduce. Aceasta conține un număr de 25 de fișiere cu o mărime de 9.23 mb și un număr de 23407 de cuvinte după aplicarea algoritmului de stemming al lui Porter.

4.1. Viteză de creare index direct + tf



```
Se creeaza indexul direct + tf... OK (0.011 secunde)
```

Figura 4.1: Timpul necesar pentru crearea indexului direct + term frequency

4.2. Viteză creare index indirect + idf



```
Se incarca vectorii asociati in memorieOK (0.354 secunde)
```

Figura 4.2: Timpul necesar pentru crearea indexului indirect + inverse document frequency

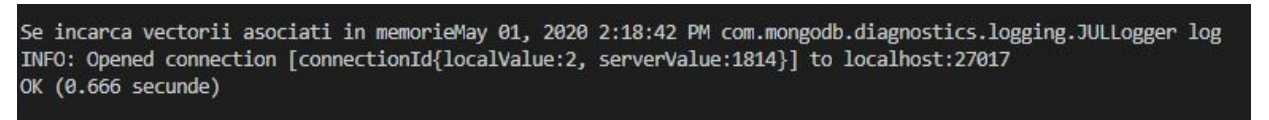
4.3. Viteză creare vectori asociați



```
Se creeaza vectorii asociati documentelor...  
OK (0.234 secunde)
```

Figura 4.3: Timpul necesar pentru crearea vectorilor asociați

4.4. Viteză încărcare vectori asociați în memorie



```
Se incarca vectorii asociati in memorieMay 01, 2020 2:18:42 PM com.mongodb.diagnostics.logging.JULLogger log  
INFO: Opened connection [connectionId{localValue:2, serverValue:1814}] to localhost:27017  
OK (0.666 secunde)
```

Figura 4.4: Timpul de încărcarea a vectorilor asociați în memorie

4.5. Viteza de căutare vectorială

```
Introduceti interogarea:
program kant immanuel asd

Rezultatele cautarii:
C:\Users\ionut\source\repos\RIW\Proiect_1\.\test-files\fie1\14.txt (relevanta 74.0%)
C:\Users\ionut\source\repos\RIW\Proiect_1\.\test-files\fie 4\25.txt (relevanta 73.0%)
C:\Users\ionut\source\repos\RIW\Proiect_1\.\test-files\fie 4\24.txt (relevanta 73.0%)
C:\Users\ionut\source\repos\RIW\Proiect_1\.\test-files\fie1\13.txt (relevanta 47.0%)
C:\Users\ionut\source\repos\RIW\Proiect_1\.\test-files\fie 2\fie 6\fie9\10.txt (relevanta 40.0%)
C:\Users\ionut\source\repos\RIW\Proiect_1\.\test-files\fie 4\19.txt (relevanta 39.0%)
C:\Users\ionut\source\repos\RIW\Proiect_1\.\test-files\fie 4\18.txt (relevanta 25.0%)
C:\Users\ionut\source\repos\RIW\Proiect_1\.\test-files\fie 2\3.txt (relevanta 25.0%)
C:\Users\ionut\source\repos\RIW\Proiect_1\.\test-files\fie 4\16.txt (relevanta 25.0%)
C:\Users\ionut\source\repos\RIW\Proiect_1\.\test-files\fie 3\1.txt (relevanta 25.0%)
C:\Users\ionut\source\repos\RIW\Proiect_1\.\test-files\fie 4\17.txt (relevanta 25.0%)
C:\Users\ionut\source\repos\RIW\Proiect_1\.\test-files\fie 2\5.txt (relevanta 25.0%)
C:\Users\ionut\source\repos\RIW\Proiect_1\.\test-files\fie 3\2.txt (relevanta 25.0%)
C:\Users\ionut\source\repos\RIW\Proiect_1\.\test-files\fie 4\15.txt (relevanta 25.0%)
OK (14 rezultate gasite in
0.008 secunde)
```

Figura 4.5: Timpul pentru căutarea vectoriala a anumitor cuvinte

Bibliografie

- 1: Alexandru Archip, Indexare și căutare - Proiect 1, ,
https://moodle.ac.tuiasi.ro/pluginfile.php/65513/mod_resource/content/1/proiect01_cautare_si_indexare.pdf
- 2: Julie Beth Lovins, The Lovins stemming algorithm, ,
<http://snowball.tartarus.org/algorithms/lovins/stemmer.html>
- 3: Martin Porter, The Porter stemming algorithm, ,
<http://snowball.tartarus.org/algorithms/porter/stemmer.html>