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

Indexare și Căutare

Proiect nr1. la disciplina
Regăsirea Informațiilor pe Web(RIW)

Student
Guțu Ion
Coordonator
Alexandru Archip

Iași, 2020

Indexare și Căutare
(Aplicație de tip consolă)

Ion Guțu

Rezumat

Lucrarea are ca scop prezentarea funcționalităților principale de care dispune aplicația creată pentru indexarea și căutarea într-un set de documente.

Această aplicație vine ca un suport pentru toți cei interesați în modul de functionare a unui motor de căutare web.

Implementarea aplicației a fost făcută în limbajul JAVA, și prezintă și o interfață cu utilizatorul prin intermediul meniului din consolă.

Introducere

Prima etapă are ca scop efectuarea unor modificări asupra funcționalităților implementate în cadrul laboratoarelor 1-4, dar și adăugarea unor noi funcționalități precum :

- Obținerea unei procesări adecvate a cuvintelor determinate in cadrul unui text;
- Studierea și implementarea unor metode de stocare a informației în baze de date non-relaționale;
- Studierea și implementarea unei metode de găsire a relevanței unui document în procesul de căutare booleană.

Scopul acestei etape presupune :

1. aducerea cuvintelor la o formă canonică cu ajutorul unui algoritm de lematizare sau stemming;
2. Posibilitatea de a stoca/extrage inexul direct/invers din o bază de date non-relațională;
3. Implementarea unei metode pentru determinarea relevanței documentelor în procesul de căutare booleană.

Capitolul 1. Implementare

Proiectul a fost implementat în limbajul JAVA. Acest limbaj dispune de câteva librării cum ar fi cazul librăriilor “JSoup” pentru procesarea fisierelor HTML și „mongodb”, care permite o operabilitate ridicată si o mobilitate în lucru cu baza de date „MongoDB” .

1.1. Interfața cu utilizatorul

Această parte a proiectului a fost implementată utilizând consola și prezintă în sine un meniu

```
1. Creaza indexul direct & Introducere in baza de date
2. Extrage index direct din  baza de date
3. Creeaza index indirect & Introducerea in baza de date
4. Extrage indexul indirect din baza de date
5. Cautare query (boolean + vectorial)
6. Iesire
---->
```

Figura 1Meniul Principal

Astfel după introducerea unei opțiuni se va executa opțiunea dorită, trebuie remarcat faptul că unele opțiuni sunt dependente de altele, în acest caz utilizatorul va fi atenționat de lipsa dependenței necesare.(vezi Capitolul 2)

1.2. Funcționalități

În aceast modul în conformitate cu .opțiunile din Figura 1 sunt prezentate principalele module ale aplicației

1.2.1. Modulul de procesare

În vederea procesării cuvintelor și a contoriza frecvența acestora este necesară parcurgerea unui director, a fiecărui fișier text și verificarea individuală a fiecărui cuvânt în parte. În acest scop a fost creată clasa „GetWords" care efectuază această procesare în următorul mod:

- Parcurgerea fiecărui caracter cu un StringBuilder
- Verificarea corectitudinii caracterului (număr sau literă) care poate forma un cuvânt
- Filtrarea contra unei liste de excepții (nume proprii sau alți termeni considerați relevanți pentru setul de documente analizat)
- Filtrarea contra unei liste de „stopwords"(acei termeni care ajută la compunerea frazelor, fără a induce informații noi (prepoziții, articole))
- Termenii de dicționar (în cazul trecerii prin filtrările precedente cuvântul se aduce la o formă canonică) se introduce în „pipeline-ul" de procesare

Algoritmii folosiți pentru aducerea unui cuvânt la o formă canonică se împart în două categorii de stemming și lematizare. Personal am ales un algoritm de stemming datorită vitezei considerabil mai mari și a unei implementări mai simple față de algoritmii de lematizare care au o complexitate mai mare, și nu oferă mereu rezultate optimiste. Procesul de stemming are rolul de a trunchia cuvintele în scopul stocării în documentul final doar a rădăcinei cuvântului, nu și a formelor derivate (de exemplu verbe cojugate, substantive la plural etc)
Algoritmul ales este Porter un algoritm destul de utilizat fiindcă folosește o abordare simplă a conflictelor și oferă rezultate destul de precise aplicate asupra unui set mare de documente.

1.2.2 Modulul de indexare

Acest modul presupune existența a două metode principale și anume **directIndex** și **indexInvers** care au ca scop crearea indexului direct, respective invers rezultat pe parcurgerea unui set de directoare.

Metoda **directIndex**:

- Parcurge iterative fiecare director
- Asupra fiecărui fișier se aplică modulu de procesare
- Se stochează un fișier cu indexul direct pentru fiecare director în parte
- Se construiește fișierul de mapare asociat setului de fișiere prelucrat

La parcurgerea unui fișier modulul de procesare returnează un HashMap cu conținut de forma <cuvant ,nr.aparitii>. În etapa următoare acest HashMap se adaugă în cel final de forma HashMap<FileName, HashMap<cuvant, nrAparitii>> - această formă este însuși forma indexului direct

$$(doc_i :< term_j, count(term_j) > | term_j \in doc_i)$$

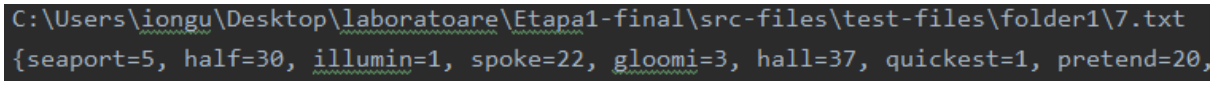

Figura 2Exemplu de stocare a indexului direct

Metoda **indexInvers**

- Pe baza indexului direct rezultat se crează indexul indirect prin modificarea structurii de salvare sub forma

$$(term_i :< doc_j, count(term_i) > |term_i \in doc_j)$$

- La fel se calculeză și fișierul de mapare respectiv indexului invers

Figura 3Exemplu de stocare a indexului invers

1.2.3 Modulul de căutare booleană

Acest modul(BooleanSearch) are la bază indexul indirect calculat în etapa precedentă și realizează o căutare printre fișierele ce conțin cuvintele introduce în interogare.

Căutarea booleană presupune existența unei interogări creată din cuvinte cheie combinate prin operatorii logici ”AND” ”OR” și ”NOT”.

Odată ce a fost introdus un query de forma

„cheie1 OPERATOR1 cheie2 OPERATOR2 cheie3”

Se trece acest query prin :

1. Filtrarea contra unei liste de excepții (nume proprii sau alți termeni considerați relevanți pentru setul de documente analizat)
2. Filtrarea contra unei liste de „stopwords”(acei termeni care ajută la compunerea frazelor, fără a induce informații noi (prepoziții, articole))
3. Termenii de dicționar (în cazul trecerii prin filtrările precedente cuvântul se aduce la o formă canonică) se introduce în „pipeline-ul” de procesare

În urma acestei procesări vor rezulta o listă cu operatori și una cu operanzi. Astfel pentru fiecare operand(cuvant) se returnează mulțimea de fișierele în care acesta se găsește, și se aplică aperatorul dintre doi operanzi pe setul de fișiere.

Pentru operatorul AND, se parcurge mulimea cu cardinalul mai mic și se adaugă la rezultat documentul curent, doar dacă acesta există în cealaltă mulțime.

La operatorul OR, se aplică principiul invers: se parcurge mulțimea cu cardinalul mai mare.

Pentru operatorul NOT nu este posibilă o optimizare datorită necomutativității acestuia.

1.2.4 Modulul de căutare vectorială(VectorialSearch)

Astfel cum căutarea booleană nu dă o estimare cantitativă a relevanței rezultatelor este necesară de a asigura o metodă matematică de calcul pentru relevanța rezultatelor, în acest scop a fost creat modulul de căutare vectorială ce implementează în sine „cosineSimilarity" similaritatea cosinus a două documente după formula

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

Figura 4Formula de calcul a similarității cosinus

Unde $\vec{d1}$ reprezinttă forma vectorială a documentului d1.

Și se obține conform algoritmului din

Figura 5

Algoritm 1 cautareVectoriala(D, q)

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

Figura 5Algoritmul Căutării Vectoriale

Principiul algoritmului presupune transformarea query-ului de căutare și a fiecărui document în parte în forma vectorială și stocare a fiecărui cuvânt sub forma {cuvant : tf(cuvant,fisier)*idf(cuvant)}.
Unde,

P1 – frecvența de apariție a cuvântului k în cadrul documentului d (în eng.: **t**erm **f**requency)

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

Figura 6 Metoda de calcul tf

P2 – frecvența inversă de apariție a cuvântului k în cadrul mulțimii de documente D (în eng.: ***i**nverse **d**ocument **f**requency*)

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

Figura 7Metoda de calcul idf

Apoi conform formulei de calcul a similarității

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

Se determină relevanța rezultatelor în ordine descrescătoare.

```
coals and flip or Russia and not
Se cauta fisierele ce satisfac cautarea

C:\Users\iongu\Desktop\laboratoare\Etapa1-final\src-files\test-files\20.txt
C:\Users\iongu\Desktop\laboratoare\Etapa1-final\src-files\test-files\folder1\1.txt
C:\Users\iongu\Desktop\laboratoare\Etapa1-final\src-files\test-files\folder1\2.txt
C:\Users\iongu\Desktop\laboratoare\Etapa1-final\src-files\test-files\folder2\12.txt
C:\Users\iongu\Desktop\laboratoare\Etapa1-final\src-files\test-files\folder2\15.txt
C:\Users\iongu\Desktop\laboratoare\Etapa1-final\src-files\test-files\folder2\17.txt
C:\Users\iongu\Desktop\laboratoare\Etapa1-final\src-files\test-files\folder2\18.txt
C:\Users\iongu\Desktop\laboratoare\Etapa1-final\src-files\test-files\folder2\19.txt
C:\Users\iongu\Desktop\laboratoare\Etapa1-final\src-files\test-files\folder1\5.txt
```

Figura 8Exemplu de căutare booleană

```
In urma procesarii vectoriale pe query. Relevanta documentelor este urmatoarea:
C:\Users\iongu\Desktop\laboratoare\Etapa1-final\src-files\test-files\folder2\19.txt->89.38%
C:\Users\iongu\Desktop\laboratoare\Etapa1-final\src-files\test-files\folder1\5.txt->88.32%
C:\Users\iongu\Desktop\laboratoare\Etapa1-final\src-files\test-files\20.txt->46.89%
C:\Users\iongu\Desktop\laboratoare\Etapa1-final\src-files\test-files\folder2\18.txt->46.89%
C:\Users\iongu\Desktop\laboratoare\Etapa1-final\src-files\test-files\folder2\17.txt->46.62%
C:\Users\iongu\Desktop\laboratoare\Etapa1-final\src-files\test-files\folder2\12.txt->45.1%
C:\Users\iongu\Desktop\laboratoare\Etapa1-final\src-files\test-files\folder2\15.txt->45.1%
C:\Users\iongu\Desktop\laboratoare\Etapa1-final\src-files\test-files\folder1\2.txt->34.2%
C:\Users\iongu\Desktop\laboratoare\Etapa1-final\src-files\test-files\folder1\1.txt->31.27%
```

Figura 9Exemplu de căutare vectorială

1.2.5 Modulul Mongo

Astfel cum generarea de fiecare data a indexilor este costisitoare din punct de vede al timpului o metodă alternativa ar presupunea stocarea acestor indexi într-o bază de date cu o scalabilitate mare(datorită dimensiunii destul de mari) și cu o mobilitate sporită(datorită diferitelor forme ale indexilor).

Astfel MongoDB este un pretendent perfect pentru stocarea datelor. Modulul „Mongo" are metode specifice pentru stocarea și preluarea a indexului direct/invers în baza de date permitând astfel restabilirea indexului invers sau direct într-un timp mai scurt.

```
public static  MongoCollection insert_direct_index(MongoDatabase database, HashMap<String, HashMap<String,Integer>> directIndex)
{
    MongoCollection direct_collection = database.getCollection( s: "directIndex");
    direct_collection.drop();
    BasicDBObject fileDB = new BasicDBObject();
    for (Map.Entry<String, HashMap<String, Integer>> file : directIndex.entrySet()) {
        fileDB.put("doc", file.getKey());
        ArrayList<DBObject> array = new ArrayList<>();
        DBObject dbo;
        for (Map.Entry<String, Integer> entry : directIndex.get(file.getKey()).entrySet()) {
            BasicDBObject document = new BasicDBObject();
            document.put("t",entry.getKey());
            document.put("c",entry.getValue());
            dbo = document;
            array.add(dbo);
        }
        fileDB.put("terms",array);
        direct_collection.insertOne(new Document(fileDB));
    }
    return direct_collection;
}
```

Figura 10Modalitate de inserare a indexului direct în MongoDB

Astfel indexul se stochează sub forma

```
"doc" : "C:\\Users\\iongu\\Desktop\\laboratoare\\Etapa1-final\\src-files\\test-files\\folder1\\5.txt",
"terms" : [
    {
        "t" : "half",
        "c" : 2
    },
    {
        "t" : "spoke",
        "c" : 2
    },
    {
        "t" : "surgeon",
        "c" : 1
    },
```

Se procedează similar și cu stocarea/extragerea indexului invers.

Capitolul 2. Mod de funcționare

Astfel meniul prezentat în Figura 1 oferă posibilitatea de a vizualiza și testa toate funcționalitățile proiectului

- Opțiunea 1:
    1. Creează indexul direct
    2. Inserează indexul direct în baza de date
- Optiunea 2(dependentă de opțiunea 1) :
    1. Preia indexul direct din baza de date
- Optiunea 3(dependentă de opțiunea 1):
    1. Creeaza indexul invers
    2. Insereaza indexul invers in baza de date
- Optiunea 4 (dependenta de optiunea 3)
    1. Preia indexul invers din baza de date
- Optiunea 5(dependent de 3)
    1. Efectueaza cautarea booleana
    2. Aplica cautarea vectoriala peste rezultatul Boolean

2.1 Optimizări

Pentru o mai bună funcționare a proiectului au fost aduse unele optimizări cu scopule de a obține o procesare mai rapidă a datelor, astfel s-a recurs la următoarele optimizări

- În modulul de procesare a cuvintelor a fost folosit un Stringbuilder pentru parcurgerea textul.
- Modulul de indexare și anume metoda de calcul a indexului invers returnează o clasă(ce are ca membri : nr. cuvinte în fișier, nr cuvinte distincte din fișier) folosită pentru calcularea similarității cosinus
- În procesul de căutare booleană se face parcurgerea mulțimilor în conformitate cu operația corespunzătoare(vezi Modulul de căutare booleană)
- În procesul de calculare a formei vectoriale a documentelor se salvează aceasta într-un fișier JSON, ce permite evitarea unor calcule suplimentare
- Odată ce a fost încărcat indexul invers din baza de date, atunci căutarea booleană nu mai necesită recalcularea indexului invers.

2.2 Necesită Optimizări

Aplicația are și punctele ei slabe care presupun :

- Apelarea recursivă de calcul al indexului direct pentru fiecare folder în parte, de dorit ar fi implementarea unei metode iterative
- Lipsa paralelizării
- Calcularea idf-ului sit tf-ului în afara modulul de indexare cee ace prespune costuri suplimentare

Concluzii

În cele din urmă pot afirma că aplicația implementată m-a ajutat să-mi aprofundez cunoștințele în modul de funcționare a unui motor de căutare.