UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

DOMENIUL: Calculatoare și tehnologia informației

SPECIALIZAREA: Tehnologia informației

Proiect RIW

Indexare și căutare

Profesor Student

Archip Alexandru Titiliuc Adrian-Valentin

Introducere

Etapa I a proiectului presupune procesarea unui set de date astfel încât să se obțină un rezultat satisfăcător in privința indexării și căutării unor cuvinte in cadrul setului de date oferit.

Pentru realizarea proiectului s-a utilizat limbajul de programare Java cu următoarele biblioteci:

1. bson-3.4.3
2. commons-io-2.4
3. jsoup-1.13.1
4. mongo-java-driver-3.12.3

Pentru o înțelegere mai bună a ce săa dezvoltat, proiectul are urmatoarea structură:

Proiect\_RIW

* Rezultat
* src
* test-files
* excepții
* stopwords

Lăsând deoparte directorul sursă se mai pot observa 4 fișiere noi care au fost folosite pentru procesarea datelor, totul plecând de la directorul cu datele de test care conține 25 de fisiere text. “stopwords” și “excepții” sunt două fișiere text care conțin cuvinte ce vor fi filtrare din fișierele de test. Tot rezultatul poate fi observat în cele din urmă în directorul “Rezultat”.

Funcționare Aplicație (Scenariu de test)

Aplicația nu necesită introducerea de la tastatură a directorului din care se vor prelua datele de test, întrucât sunt aceleași de fiecare dată, deci a fost implementat direct în aplicație un string care conține această cale.

Se preiau fișierele text din directorul “test-files” și se creează o buclă în interiorul căreia se va procesa fiecare fișier după cum urmează:

1. Se extrag cuvintele și se scriu într-un fișier cu numele de forma In<nume\_fișier\_inițial>, fișier care servește drept input pentru crearea index-ului direct lucru ce face procesarea fișierelor HTML posibilă.
2. Se face o parsare a documentului creat cu numele In<număr\_fișier> de unde se preia fiecare cuvânt și se verifică dacă acesta aparține listei cu “stopwords”, dacă nu se regăsește în acea listă de cuvinte se va trece la pasul următor care presupune aplicarea algoritmului Porter peste acel cuvânt care presupune obținerea formei canonice. După ce s-a obținut forma canonică se formează index-ul direct și se adaugă într-un HashMap de forma <string, integer> unde string va conține documentul și cuvântul, iar integer va fi pentru numărul de apariții al acelui cuvânt. Dacă respectivul cuvânt există deja în lista mai sus menționată atunci se va incrementa valoarea care reprezintă numărul de apariții. În final se va scrie un fișier rezultat cu forma Output<număr\_fișier> care va conține indecșii corespunzători fișierului procesat care sunt de forma:

<cale\_fișier\nume\_fișier\_inițial, cuvânt, număr\_apariții\_cuvânt>

1. La acest pas se procesează fișierele obținute la pasul anterior și se creează indexul direct care este mai apoi scris în fișierul “FinalOutput”. Index-ul indirect și final în acest caz va avea forma:

<cuvânt, cale\_fișier\nume\_fișier\_inițial, număr\_apariții\_cuvânt>

În forma prezentată mai sus, în cadrul fiecărui index, după cuvânt vor apărea toate fișierele, după fiecare fișier fiind numărul de apariții al acestui cuvânt.

1. După ce s-a finalizat procesarea tuturor documentelor, iar fișierul “FinalOutput” a fost scris se va apela o funcție care are rolul de a calcula IDF-ul pentru fiecare cuvânt, rezultatul final putând fi observat în fișierul “FinalIDF”.
2. La acest pas s-a implementat o funcție care calculează într-un final vectorul asociat fiecărui document sub forma:

<cuvânt, document, tf/idf>

Concluzii

Pe parcursul dezvoltării proiectului s-a încercat utilizarea unei baze de date nerelațională, însă acest lucru a fost fără succes. De asemenea, nu am reușit să implementez căutarea vectorială, însă am implementat o funcție care ar trebui să calculeze distanța cosinus dintre un cuvânt și query-ul asociat acelui cuvânt.