UNIVERSITÀ DEGLI STUDI ROMA TRE

Ingegneria dei Dati

Homework 2

Gaglione Giulia

Matricola 559057

Anno Accademico 2024/2025

Apache Lucene

Github URL: `https://github.com/giug2/IDD-Homeworks.git`

Lo scopo di questo progetto è quello di realizzare un programma Java che si occupi di indicizzare e ricercare efficientemente i contenuti all'interno di articoli scientifici in formato .html, situati in una directory definita, all_htmls.
La gestione della creazione e interrogazione degli indici viene effettuata utilizzando *Apache Lucene*, una potente libreria open-source per il text-searching e l'indicizzazione.
Lucene permette di creare indici personalizzati che rendono più rapide e mirate le ricerche all'interno di grandi quantità di dati testuali. In questo progetto, Lucene viene impiegato per costruire indici capaci di catalogare ogni articolo in base a specifici campi chiave, come ad esempio il titolo, gli autori e il contenuto.
La definizione di questi campi permette di strutturare l'indice in modo che le ricerche possano essere affinate e targettizzate, facilitando così l'utente nella ricerca di informazioni precise e rilevanti in base ai criteri selezionati.
Le query possono includere la ricerca per titolo, per autore o per contenuto. Il programma Java è in grado di leggere la query direttamente da console, di interrogare gli indici creati e di stampare poi i risultati.

1.1 Il codice

Di seguito un'analisi specifica per ogni classe Java.

1.1.1 LuceneHW

È il punto di ingresso del programma Java.

```
public class LuceneHW {

    1 usage
    private static Indexer creator = new Indexer();
    1 usage
    private static Searcher searcher = new Searcher();

    public static void main(String[] args) {

        creator.createIndex();
        new Stats().statsIndex();

        // Inizio loop di ricerche tramite prompt
        while(true){
            searcher.searchIndex();

            Scanner scanner = new Scanner(System.in);
            System.out.print("Vuoi continuare la ricerca? Y/N:  ");
            String exit = scanner.nextLine();

            if (exit.equalsIgnoreCase( anotherString: "N")) {
                System.out.println("Fine ricerca.");
                break;
            }
        }
    }
}
```

All'inizio della funzione main, viene chiamato il metodo *createIndex()* sull'oggetto *creator*, che è un'istanza della classe *Indexer*. Questo metodo è responsabile della creazione dell'indice.

Subito dopo la creazione dell'indice, viene creata un'istanza temporanea della classe *Stats* e viene chiamato il metodo *statsIndex()* che si occupa di fornire statistiche sull'indicizzazione.

Infine, il programma entra in un ciclo *while (true)* che permette all'utente di eseguire ricerche continue sull'indice.

1.1.2 Indexer

La classe *Indexer* è progettata per creare un indice Lucene basato su documenti .html memorizzati in una specifica directory.
La mappa *perFieldAnalyzers* viene creata per specificare diversi Analyzer per ogni campo.
analyzerCustom è un CustomAnalyzer creato per analizzare i campi title e authors. La configurazione include:

- *WhitespaceTokenizerFactory*: divide il testo in token separati da spazi.
- *LowerCaseFilterFactory*: converte tutto il testo in minuscolo per uniformare le ricerche.
- *WordDelimiterGraphFilterFactory*: filtra i termini per dividerli ulteriormente in base a delimitatori comuni, come trattini o caratteri speciali.

```
Map<String, Analyzer> perFieldAnalyzers = new HashMap<>();

// Analyzer personalizzato
Analyzer analyzerCustom = CustomAnalyzer.builder()
        .withTokenizer(WhitespaceTokenizerFactory.class)
        .addTokenFilter(LowerCaseFilterFactory.class)
        .addTokenFilter(WordDelimiterGraphFilterFactory.class)
        .build();
```

I campi *title* e *authors* utilizzano *analyzerCustom*, mentre il campo *content* usa *StandardAnalyzer* con una lista di parole di stop definite nella classe *Stopwords*. Se un campo non ha uno specifico *Analyzer* nella mappa, userà l'*EnglishAnalyzer* impostato di default.

```
perFieldAnalyzers.put("title", analyzerCustom);
perFieldAnalyzers.put("authors", analyzerCustom);
perFieldAnalyzers.put("content", new StandardAnalyzer(new Stopwords().getStopWords()));

Analyzer perFieldAnalyzer = new PerFieldAnalyzerWrapper(new EnglishAnalyzer(), perFieldAnalyzers);
```

1.1.3 Docs

Il codice della classe *Docs* si occupa della lettura, analisi e conversione di file .html in documenti indicizzabili da Apache Lucene. Questa classe è progettata per estrarre informazioni come titolo, autori e contenuto da file .html situati in una directory specifica, creando un elenco di documenti Lucene pronti per l'indicizzazione.

```
// Estrai il titolo dall'elemento
Element titleElem = jsoupDoc.selectFirst( cssQuery: "h1.ltx_title.ltx_title_document");
String title = (titleElem != null) ? titleElem.text() : "Titolo non trovato";

// Estrai gli autori dall'elemento
Element authorElem = jsoupDoc.selectFirst( cssQuery: "div.ltx_authors");
List<String> authors = extractAuthors(authorElem);

// Estrai il contenuto testuale rimuovendo i tag HTML
String content = jsoupDoc.text();
```

In particolare, la funzione *extractAuthors* si occupa della pulizia e gestione dei nomi degli autori.

1.1.4 Stats

Il metodo *statsIndex* della classe *Stats* fornisce una panoramica dell'indice Lucene analizzando il numero di documenti indicizzati e il conteggio dei termini per ciascun campo.

1.1.5 Searcher

La classe *Searcher* consente di effettuare ricerche sui documenti indicizzati con Lucene, utilizzando una combinazione di campi, titolo e autori, oppure un singolo campo scelto dall'utente. Oltre a effettuare la ricerca, la classe calcola le metriche di precisione e richiamo, fondamentali per valutare l'efficacia della ricerca.

1.1.6 Stopwords

La classe *Stopwords* è progettata per creare e gestire un insieme di parole comuni che possono essere ignorate durante il processo di indicizzazione e ricerca per migliorare l'efficienza e la rilevanza dei risultati.

1.2 Indicizzazione

Dall'avvio dell'indicizzazione fino alla fine di essa passano esattamente 973006 ms, ossia circa 16 minuti.

```
Tempo impiegato per l'indicizzazione: 973006 ms
```

Durante l'indicizzazione vengono ritrovati tutti i documenti .html all'interno della directory all_htmls.

```
Numero di documenti indicizzati: 9372
```

1.3 Analyzer utilizzati

1.3.1 CustomAnalyzer

Il CustomAnalyzer è configurato con il WhitespaceTokenizer e due TokenFilters:

- *WhitespaceTokenizerFactory*: tokenizza il testo separando i termini in base agli spazi bianchi. Questo significa che ogni parola o gruppo di parole separato da uno spazio diventa un token. Non vengono applicati altri separatori più complessi come la punteggiatura o caratteri speciali.
- *LowerCaseFilterFactory*: questo filtro converte tutti i termini in minuscolo, uniformando il testo e garantendo che le ricerche non facciano distinzione tra maiuscole e minuscole.
- *WordDelimiterGraphFilterFactory*: questo filtro viene utilizzato per trattare i delimitatori di parole, come il trattino , il sottolineato _, e i caratteri speciali, trattandoli come separatori di parole o "unificando" le parole separate da questi simboli.

L'analyzer è utilizzato per i *titoli* e per gli *autori*.

1.3.2 StandardAnalyzer

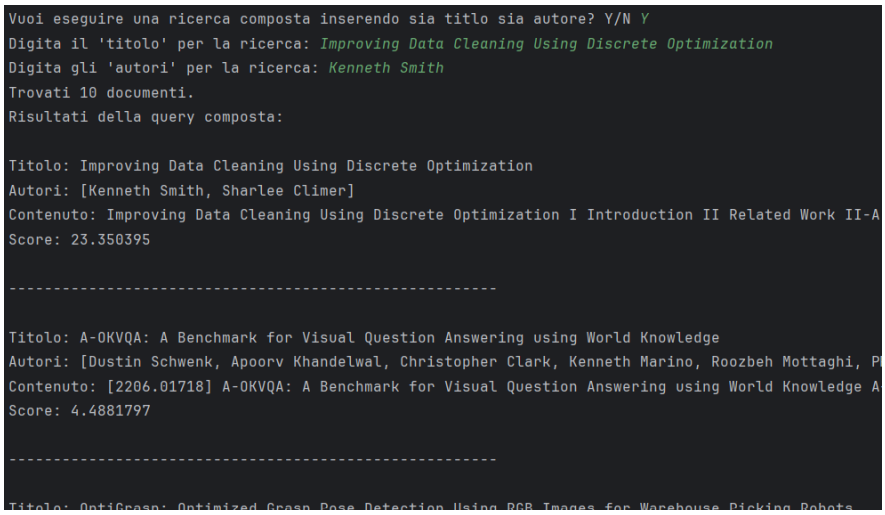
Per il campo *content*, viene utilizzato lo StandardAnalyzer con un filtro di stopwords personalizzato, ovvero un set di parole comuni che vengono ignorate durante la ricerca. L'uso di StandardAnalyzer è più adatto per l'analisi del contenuto di un documento, poiché il contenuto potrebbe contenere una varietà di formati e di parole comuni che dovrebbero essere filtrate per evitare risultati inutili o imprecisi.

1.4 Query di test

Sono state poi eseguite query di test.

1.4.1 Query combinata titolo e autore

Tempo di risposta: 253944.8318 ms



1.4.2 Query su titolo generale

Tempo di risposta: 14418.9219 ms

```
Su quale campo vuoi eseguire la tua ricerca: title
Digita la tua query: Data Cleaning
Numero di risultati: 10

Trovati 10 documenti.
Risultati della query:

Titolo: Cleaning data with Swipe
Autori: [Toon Boeckling, Antoon Bronselaer]
Contenuto: Cleaning data with Swipe 1 Introduction The Llunatic Chase algorithm Single-path Chase trees with Swipe 2 Re
Score: 5.587655

----------------------------------------------------------

Titolo: Improving Data Cleaning Using Discrete Optimization
Autori: [Kenneth Smith, Sharlee Climer]
Contenuto: Improving Data Cleaning Using Discrete Optimization I Introduction II Related Work II-A Greedy II-B RowCol
```

1.4.3 Query su autore generale

Tempo di risposta: 123341.5686 ms

```
Vuoi eseguire una ricerca composta inserendo sia titlo sia autore? Y/N N
Su quale campo vuoi eseguire la tua ricerca: author
Campo non valido!
Su quale campo vuoi eseguire la tua ricerca: authors
Digita la tua query: Antoon
Numero di risultati: 1

Trovati 1 documenti.
Risultati della query:

Titolo: Cleaning data with Swipe
Autori: [Toon Boeckling, Antoon Bronselaer]
Contenuto: Cleaning data with Swipe 1 Introduction The Llunatic Chase algorithm Single-path Chase trees
Score: 5.8158674

----------------------------------------------------------
```

1.4.4 Query su titolo specifico

Tempo di risposta: 6193.3485 ms

```
Su quale campo vuoi eseguire la tua ricerca: title
Digita la tua query: Cleaning data with Swipe
Numero di risultati: 10

Trovati 10 documenti.
Risultati della query:

Titolo: Cleaning data with Swipe
Autori: [Toon Boeckling, Antoon Bronselaer]
Contenuto: Cleaning data with Swipe 1 Introduction The Llunatic Chase algorithm Single-path Chase tr
Score: 12.063755

-----------------------------------------------------

Titolo: Improving Data Cleaning Using Discrete Optimization
Autori: [Kenneth Smith, Sharlee Climer]
Contenuto: Improving Data Cleaning Using Discrete Optimization I Introduction II Related Work II-A G
```

1.4.5 Query su autore specifico

Tempo di risposta: 7371.2653 ms

```
Vuoi eseguire una ricerca composta inserendo sia titlo sia autore? Y/N N
Su quale campo vuoi eseguire la tua ricerca: authors
Digita la tua query: Kenneth Smith
Numero di risultati: 10

Trovati 10 documenti.
Risultati della query:

Titolo: Improving Data Cleaning Using Discrete Optimization
Autori: [Kenneth Smith, Sharlee Climer]
Contenuto: Improving Data Cleaning Using Discrete Optimization I Introduction II Related Work II-A Greedy II-
Score: 8.160881

-----------------------------------------------------

Titolo: One-Shot Federated Learning with Neuromorphic Processors
Autori: [Kenneth Stewart, Yanqi Gu]
Contenuto: [2011.01813] One-Shot Federated Learning with Neuromorphic Processors One-Shot Federated Learning
```

1.5 Conclusioni

In conclusione, la personalizzazione dell'analisi dei campi, resa possibile grazie all'uso di strumenti avanzati come l'Analyzer, rappresenta un elemento cruciale per ottimizzare l'interazione dell'utente con il sistema.

Inoltre, l'uso di Analyzer non solo facilita la gestione dei dati, ma contribuisce anche a garantire una maggiore precisione e affidabilità nelle elaborazioni, riducendo il rischio di errori e aumentando l'efficienza del sistema.