Information Retrieval

Assignment 2

Name	ID	Group
Youssef Mohamed Atya	20210484	S4
Abdelrhman Mostafa Hessain	20210210	S6
mostafa saeed mowoad	20220336	ALL
محمد ايمن السيد محمد	20210569	S6
Doaa Mahdy Mohamed	20210128	S2
Ahmed Mohamed Ahmed Sadek	20210029	ALL

1- Crawler

1- **Seed URLs**
   a. The crawler starts with a predefined list of seed URLs added to a queue.

2- **Data Structures**
   a. A queue (FIFO) to manage the list of pages to visit.
   b. A hash set to keep track of visited URLs and avoid duplicates.

3- **Crawl Process**
   a. While the queue is not empty and the visited set size is below the limit, take the next URL.
   b. Fetch and parse the HTML content using Jsoup.
   c. Extract plain text from paragraphs in the main content area.
   d. Save the text to a file named after the URL's last segment.

4- **Link Extraction**
   a. Scan all <a> tags to find Wikipedia links matching a specific pattern.
   b. Add new links to the queue if not visited and not already in the queue.

5- **Limiting Growth**
   a. The size of visited pages plus the queue is capped to avoid exceeding the maximum set by the crawler.

2- Inverted Index

An inverted index matches each unique term to the documents in which it appears. The process begins by reading each document line by line. Each line is tokenized into words removing punctuation and converting text to lowercase. Stopwords (e.g., “the,” “a,” “and”) are filtered out to reduce noise.

For every remaining word, the system checks if it already exists in the dictionary. If not, it creates a new dictionary entry; if yes, it updates the corresponding posting list. A posting list holds records of document IDs and how frequently that term appears in each document. When a term is encountered for the first time in a new document, an entry is added to the posting list. Each subsequent occurrence increments a counter rather than creating new entries. Over time, this creates a consolidated structure that quickly returns all documents containing a given term.

3- TF-IDF and Cosine Similarity

1. **Term Frequency (TF)**
    - Counts how many times a term appears in a document.
    - Uses the formula:
      $TF(t,d) = 1 + \log_{10}(\text{count}(t) \text{ in } d)$
2. **Inverse Document Frequency (IDF)**
    - Highlights the importance of rare terms across all documents.
    - Formula:
      $IDF(t) = \log_{10}(\text{(Number of docs containing t) / (Total number of docs)})$
3. **TF-IDF**
    - Combines TF and IDF to rank term relevance:
      $TF\text{-}IDF(t,d) = TF(t,d) \times IDF(t)$
    - Each term’s final score is stored per document.

4. **Cosine Similarity**
   - Measures similarity between two vectors (query vs. document).
   - Steps involve computing:
     • Dot product of the vectors.
     • Norm (length) of each vector.
     • Final similarity = (Dot product) / (Norm of query × Norm of document).

4- User Manual

1. **Compile and Run the Program**
   - Make sure you have a Java environment set up.
   - Navigate to the invertedIndex folder where Test.java is located.
   - Compile and run the Test.java file from the command line or your IDE.
2. **How It Works**
   - The program first gathers data by calling WikiCrawler.
   - It stores the fetched text files in the Path folder.
   - The Index5 class creates an inverted index, while TFIDF computes term frequencies for ranking.
3. **Entering a Query**
   - When prompted, type any search terms and press Enter.
   - The program will display the top results based on cosine similarity.
   - Type ‘exit’ (then press Enter) to quit the application.