Mini Search Engine Design

Team #2

Semester

Mohamed Shawky
SEC:2, BN:16

Remonda Talaat
SEC:1, BN:20

Evram Youssef
SEC:1, BN:9

Mahmoud Adas
SEC:2, BN:21

June 2, 2020

Contents

1 Introduction

This document gives you a slight walk through the code and introduces you briefly to the used techniques and algorithms.

The backend uses Java-Spring framework, which requires the classes that want to access a component (e.g. database) to be a component itself, this is why you will notice the annotation `@Component` used in many classes around.

2 Algorithms

This section describes briefly some techniques and methods used along the project.

2.1 Crawling

Crawling is composed of the following classes:

- `UrlsStore`: Singleton, has a priority queue full of links.

  If the server closes, `UrlsStore` will save the queue in db and load them the next time it starts.

  It loads the crawling seed if the db is empty.

  The priority of a url is inversely proportional to its url based last access time. The more you visit some website, the less priority any links of it gets. The goal is to favour new websites, instead of going deeper into one website.

- `DocumentsStore`: Singleton in a separate thread. Has a queue of documents to store at the db when the db is not locked.

- `Crawler`: Has many threads, each pulls `UrlsStore`'s queue for a url.

  Once it has a url, it fetches the document, ignores the document if not html, otherwise extract all urls and image links in it.

  Stores the image links in db, puts the urls into `UrlsStore`'s queue and puts the fetched document into `DocumentsStore`'s queue.

  It goes like that forever.

2.2 Indexing

There is only one indexer thread. Simply it repeats the following forever:

1. Sleeps for couple of seconds.
2. Fetches all non-indexed documents.
3. Extracts unique and most important keywords from their contents.
4. Insert the keywords into `keywords` table in db.
5. Store the relationship between each keyword and the fetched document in `keyword_document` table in db.

2.3 Queries Handling

2.3.1 Phrase Processor

1. Parses the phrases in quotes.
2. Remove them from the original query.
3. Execute simple SQL query using "LIKE" syntax to match the keywords in any content in all files.
4. Return the files.

2.3.2 Query Processor

Works after phrases are removed by `PhraseProcessor`.

1. Extract the keywords.
2. Stem them.
3. Search for the keywords in `keywords` table.
4. Apply SQL joins with `keyword_document` and `documents` table to get the matching documents.
5. Send results to relevance ranking.

2.4 Ranking

2.4.1 Popularity Ranking

A separate thread that calculate popularity ranking for crawled urls in the database and stores it back in the database. The used ranking algorithm is `PageRank`. It recursively does the following:

- Read urls from database.
- Reset all ranks to $1/\#urls$.
- Do the following for 5 iterations, for each url:
  - Calculate the incoming rank of the url.
  - Calculate the outgoing rank of each incoming url.
  - Compute PageRank.

2.4.2 Relevance Ranking

Called by Query Processor to sort the query results. Depends on four criteria:

- Popularity Rank (from database).
- Term Frequency - Inverse Document Frequency *(TF-IDF)*.
- Geographic Location of client.
- Recent Publish Dates.

The following procedure happens:

- query results are sorted according to each criterion, independently.
- based on each order, each result is given a score of 0, 1, ... etc.
- these scores are added up to form the final score of each result.
- results are then sorted descendingly according to the final score.