COMP4321 PROJECT

WEB-BASED SEARCH ENGINE

<u>FINAL REPORT</u>

GROUP 43

PARTICIPANTS:

YUEN MAN HIM BOSCO

TSANG HING KI JIMMY

CHAN CHAR-JU JULIUS

12 / 4 / 2024

Introduction

This report presents a comprehensive overview of our web search engine project. It highlights the architecture and implementation details for efficient crawling, indexing, retrieval of web content, and search engine interface. The search engine is structured into several modular components, with each being assigned to specific tasks including web crawling, content extraction, indexing, and data storage management. By adopting this approach, our search engine aims to deliver a detailed and optimized user experience on web searching.

System Architecture

The search engine's system architecture is composed of several key components:

Spider (Crawler)

The **Spider** class is the web crawler of our search engine. It is responsible for managing a queue of URLs to visit, extracting content from these URLs, and passing the content to the indexer for processing. The class also handles the storage and retrieval of web pages using a JDBM database, ensuring that each page is visited only once, and updates are managed appropriately. After crawling, data will not be processed just by crawling, these data will be passed to the indexer for further processing and preparation for being searched instead.

Indexer

Our content extraction and indexing component – **Indexer** class, plays a crucial role in processing the content obtained from the Spider class. By capturing and structuring relevant data elements, the Indexer processes the topic and body text from each web page by removing stop-words and applying stemming algorithms to reduce words to their base forms. The processed data is then stored in a database.

Search Engine

The **Search Engine** class is a key component in our web search engine, responsible for processing user queries and ranking pages based on the indexed content. This class links the raw data management by Spider and Indexer with user interactions, which is the core of our project.

Interface

The **Interface** is incorporated into our search engine to provide a simple user-friendly platform for users to perform searching based on word parsing. Due to time constraints, we could only provide a basic interface for asking user to input queries and give out searching results.

File Structures

Spider

- **Pages**: A data structure HTree is used to store *PageInfo* objects, which encapsulates all relevant data for a web page, including its URL, title, body content, metadata (like last modification date), and links to other pages. The key is a unique page identifier, and the value is the serialized *PageInfo* object.

- **PageId**: This HTree maps URLs to their corresponding page identifiers. It is used to check quickly whether a page has already been crawled and indexed, thus preventing duplicate processing.

Indexer

- **TitleIndex** and **BodyIndex**: These HTrees store the indexing information for words found in the title and body of web pages, respectively. Each tree maps a word identifier to a *PostingList*, which contains frequency data and page references where the word appears. This structure allows quick lookups during search queries to determine which pages are relevant based on the indexed terms.

- **WordToIdIndex** and **IdToWordIndex**: To efficiently manage and query the words, the Indexer uses two maps. WordToIdIndex maps each unique word to a unique identifier (word ID), while IdToWordIndex provides the reverse mapping. This bi-directional mapping reduces storage requirements and speeds up both indexing and query operations.

- **PageIndex**: This HTree records metadata about each indexed page, primarily to handle updates efficiently. It stores timestamps or other relevant metadata to quickly ascertain if a page needs re-indexing due to content changes.

Workflow of the Search Engine

Initialization

When the web interface (InputQuery.html) is loaded, the Spider class is triggered to start crawling. It begins from a predefined root URL and is set to crawl up to 300 pages. This process involves fetching each webpage, extracting its content, and then passing this data to the Indexer.

Crawling and Indexing

The Indexer processes each page to extract and index keywords from the page's title and body. These keywords are stemmed and stopwords are removed to enhance the search effectiveness. The indexed keywords are stored in separate JDBM HTrees (titleIndex and bodyIndex) for the title and body of the pages, respectively. Additional page information such as page ID, URL, last modification date, page size, and links (both parent and child) are stored in another HTree called Pages.

User Interaction

Users interact with the system via the web interface, where they can enter queries into a text box and submit them using a search button. The query is then processed by the SearchEngine class.

Query Processing and Retrieval

The SearchEngine takes the user's query, tokenizes it, and processes it through stemming and stopword removal. It then retrieves the indexed pages and computes a relevance score for each page based on cosine similarity between the query vector and the document vectors stored in the HTrees.

Ranking and Results Display

The pages are ranked according to their relevance scores, and the top 50 pages are selected. These top pages are then displayed on the user's interface, showing the most relevant pages based on the query.

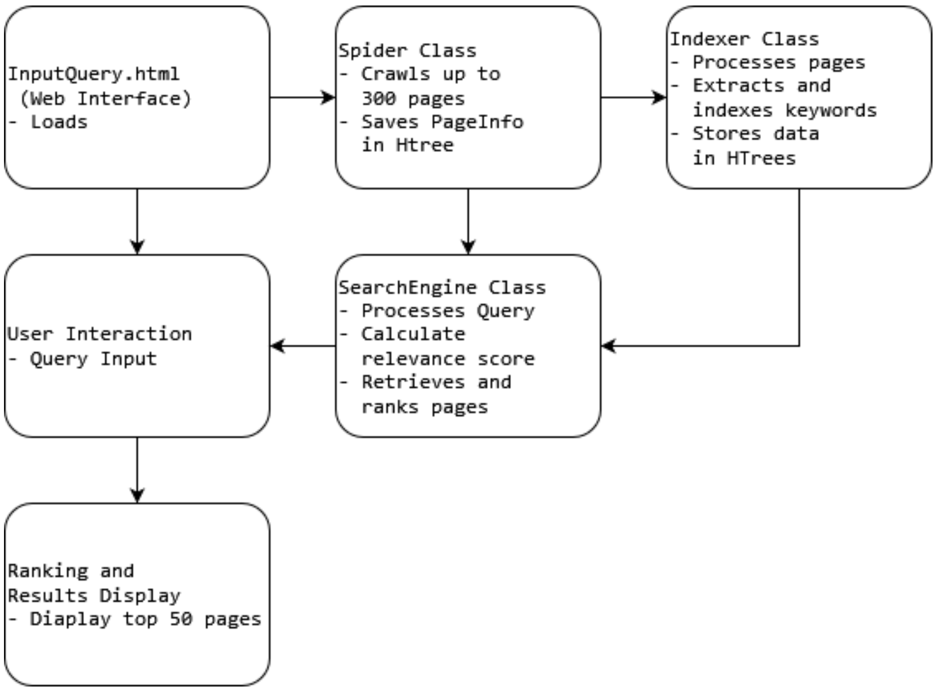


Figure 1. Diagram of the workflow

Algorithm Design

Web Crawling and Content Extraction (BFS Algorithm)

- **Breadth-First Search (BFS) on Root Link:** The *BFS_OnRootLink* method implements a breadth-first search algorithm starting from a given root URL. This method handles the core functionality of the web crawler.
- **Queue Management:** URLs to be crawled are managed using a queue. The BFS continues until the queue is empty or the maximum crawl size is reached.

- **Page Processing:** For each URL, the Spider checks if it has already been crawled. If not, it uses the Extractor class to fetch the page, extract links, the title, the body, and other metadata. New pages are indexed, and their information is stored in the HTrees.
- **Link Extraction and Looping:** Links found in the current page are added to the queue for future crawling. If a link already exists in the database but has been updated (checked by last modification date), the existing entry is updated.
- **Keyword Indexing:** Extracted titles and bodies are processed to index their keywords. This involves splitting the text into words, converting to lowercase, and then indexing these words using the Indexer.
- **Parent Link Tracking:** For each extracted link, it checks if the link is already in the database. If so, the parent link information is updated. If not, it records the current link as the parent for future reference when this link is eventually crawled.

Indexing Process

- **Stemming and Stopword Removal:** The indexing process involves cleaning the text by removing stopwords and applying stemming to reduce words to their base forms. This is facilitated by the StopStem class.
- **n-Gram Generation:** It generates stemmed n-grams (combinations of n consecutive words) to be indexed. This helps in capturing phrasal expressions which can be important for understanding the context or improving search quality.
- **Index Words:** Each word and n-gram is processed to determine its unique ID, update the posting lists (which track document IDs and frequencies for each word), and update the respective HTrees. The index is updated by either adding a new entry or updating an existing one.
- **Unique Word IDs:** Manages the assignment of unique IDs to words. If a word does not already have an ID, it is assigned a new one, and the mappings are updated in *wordToIdIndex* and *idToWordIndex*.

Retrieval Function

- **Tokenization and Preprocessing:** The function processes a raw query string by first handling exact phrase matches, which it detects by looking for text enclosed within quotes. After identifying these phrases, it splits the query into individual terms and phrases. Each term then undergoes stemming and stopword removal to refine the search terms into their core components. Finally, the processed tokens are converted into their corresponding word IDs, with the indexer maintaining a consistent mapping between terms and their IDs to ensure accurate and efficient search query handling.

- **Document and Query Vectorization:** Converts the processed query and the contents of each document into vectors of terms weighted by TF-IDF values.
- **Weighted TF Calculation:** The *PageInfo* class includes a method *getTF* which computes a term's frequency in the document. It gives a higher weight to terms found in the title (weight * 4) compared to those found in the body. This is a way to favour matches found in titles over those in the body.
- **Similarity Calculation:** Calculates the cosine similarity between the query vector and each document vector. The similarity score determines how relevant a document is to the given query.
- **Ranking:** Documents are ranked based on their cosine similarity scores in descending order.

Installation Procedure (*)

For guidelines to install and run our search engine, please read through the readme.txt file thoroughly for obtaining the best optimized results.

Bonus Features

What our search engine system can achieve beyond requirements, is the additional **"get similar pages"** function. It represents a significant advancement in enhancing the search experience for users. For each result page, a "Get similar pages" option is provided, users can indicate their preferences by clicking the button, and our search engine will dynamically generate a list of similar webpages based on the respective page's **top 5 most frequent keywords**.

Also named as **relevance feedback**, users can actively participate in shaping their search results, thus leading to a more accurate outcome, enhancing the search accuracy and quality. Moreover, the "Get similar pages" option not only save users time but also ensures that they discover and explore a wide range of relevant content that aligns with their specific interests and needs, which would give a more user-centric search experience.

Also, we implement "I am feeling lucky" button to directly get to the highest score page just like in the google search engine.

And keyword searching, it can select the keywords he/she is interested in, and then submit them as a vector-space query to your search engine.

Test Results

By integrating the above-mentioned features, functionalities, and algorithms, here are our search engine's test results:

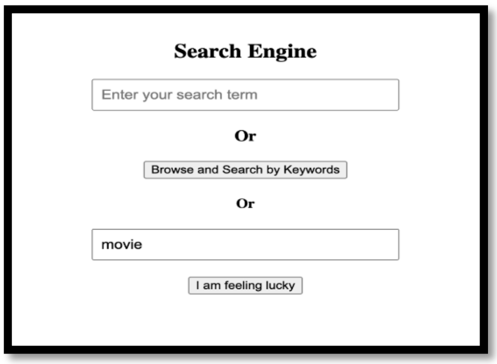
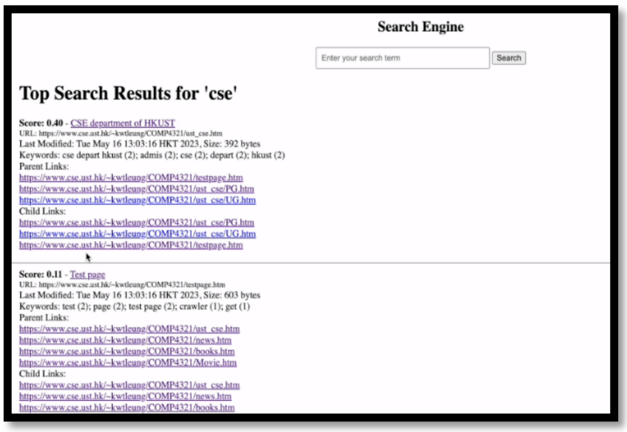


Figure 3. Initial user interface

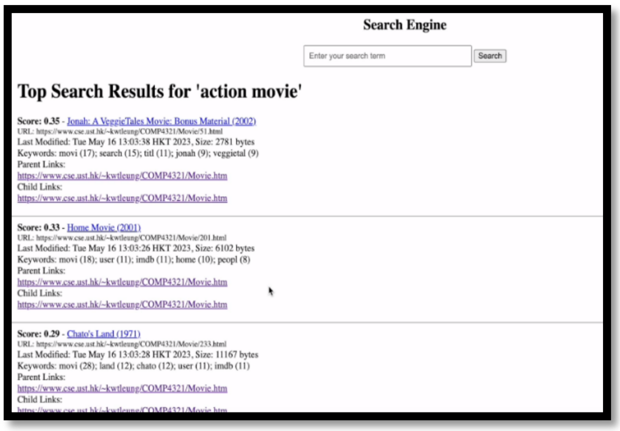


Top Search Results for 'cse'

Score: 0.40 - CSE department of HKUST
URL: https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse.htm
Last Modified: Tue May 16 13:03:16 HKT 2023, Size: 392 bytes
Keywords: cse depart hkust (2); admis (2); cse (2); depart (2); hkust (2)
Parent Links:
https://www.cse.ust.hk/~kwtleung/COMP4321/testpage.htm
https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse/PG.htm
https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse/UG.htm
Child Links:
https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse/PG.htm
https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse/UG.htm
https://www.cse.ust.hk/~kwtleung/COMP4321/testpage.htm

Score: 0.11 - Test page
URL: https://www.cse.ust.hk/~kwtleung/COMP4321/testpage.htm
Last Modified: Tue May 16 13:03:16 HKT 2023, Size: 603 bytes
Keywords: test (2); page (2); test page (2); crawler (1); get (1)
Parent Links:
https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse.htm
https://www.cse.ust.hk/~kwtleung/COMP4321/news.htm
https://www.cse.ust.hk/~kwtleung/COMP4321/books.htm
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie.htm
Child Links:
https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse.htm
https://www.cse.ust.hk/~kwtleung/COMP4321/news.htm
https://www.cse.ust.hk/~kwtleung/COMP4321/books.htm

Figure 4. Search results (keyword search)



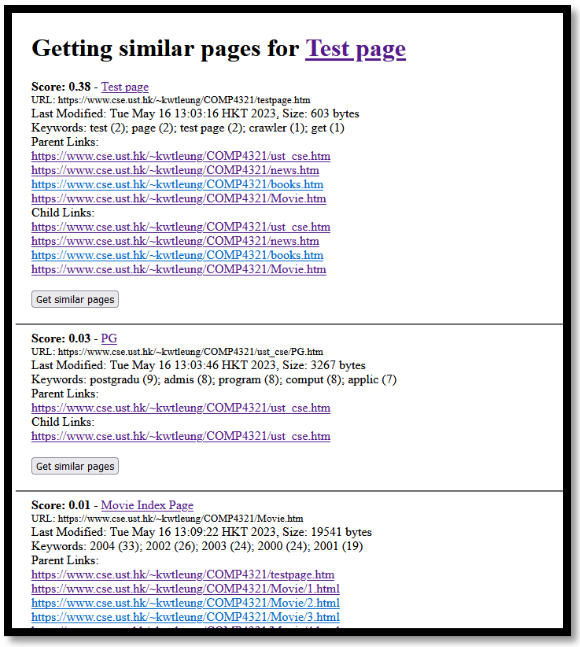
Top Search Results for 'action movie'

Score: 0.35 - Jonah: A VeggieTales Movie: Bonus Material (2002)
URL: https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/51.html
Last Modified: Tue May 16 13:03:38 HKT 2023, Size: 2781 bytes
Keywords: movi (17); search (15); titl (11); jonah (9); veggietal (9)
Parent Links:
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie.htm
Child Links:
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie.htm

Score: 0.33 - Home Movie (2001)
URL: https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/201.html
Last Modified: Tue May 16 13:03:26 HKT 2023, Size: 6102 bytes
Keywords: movi (18); user (11); imdb (11); home (10); peopl (8)
Parent Links:
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie.htm
Child Links:
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie.htm

Score: 0.29 - Chato's Land (1971)
URL: https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/233.html
Last Modified: Tue May 16 13:03:28 HKT 2023, Size: 11167 bytes
Keywords: movi (28); land (12); chato (12); user (11); imdb (11)
Parent Links:
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie.htm
Child Links:

Figure 2. Search results (phrase search)



Getting similar pages for Test page

Score: 0.38 - Test page
URL: https://www.cse.ust.hk/~kwtleung/COMP4321/testpage.htm
Last Modified: Tue May 16 13:03:16 HKT 2023, Size: 603 bytes
Keywords: test (2); page (2); test page (2); crawler (1); get (1)
Parent Links:
https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse.htm
https://www.cse.ust.hk/~kwtleung/COMP4321/news.htm
https://www.cse.ust.hk/~kwtleung/COMP4321/books.htm
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie.htm
Child Links:
https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse.htm
https://www.cse.ust.hk/~kwtleung/COMP4321/news.htm
https://www.cse.ust.hk/~kwtleung/COMP4321/books.htm
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie.htm

Score: 0.03 - PG
URL: https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse/PG.htm
Last Modified: Tue May 16 13:03:46 HKT 2023, Size: 3267 bytes
Keywords: postgradu (9); admis (8); program (8); comput (8); applic (7)
Parent Links:
https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse.htm
Child Links:
https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse.htm

Score: 0.01 - Movie Index Page
URL: https://www.cse.ust.hk/~kwtleung/COMP4321/Movie.htm
Last Modified: Tue May 16 13:09:22 HKT 2023, Size: 19541 bytes
Keywords: 2004 (33); 2002 (26); 2003 (24); 2000 (24); 2001 (19)
Parent Links:
https://www.cse.ust.hk/~kwtleung/COMP4321/testpage.htm
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/1.html
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/2.html
https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/3.html

Figure 5. Get similar pages (bonus feature)

Conclusion

Our Search Engine system aims to provide a sharp searching performance and a satisfying user experience. Although it demonstrates several strengths on its architecture and implementation, there are potential improvements on interface functionality, scalability, and user experience.

Strengths

Our system incorporates a well-designed Spider and Indexer to crawl and index web pages effectively. Due to the use of BFS algorithm, it ensures the coverage and data retrieval to be efficient, thus providing faster search results. It also ensures the search accuracy of by using both cosine similarity and weighted TF calculations as ranking algorithms.

Moreover, our search engine improves search quality by using n-Gram generation to help capture phrases for better text implication, as a result giving users a more comfortable searching experience.

Weaknesses

Again, due to time constraints, the interface of our search engine provides only basic functionality. Also, the current system may face scalability issues when handling large volumes of web pages and user queries. This is currently constrained by the project specifications, and scaling our search engine system to accommodate increased traffic and data volumes would be a key consideration for future iterations.

If we could re-implement the entire system, we would prioritize the following features and improvements:

User Experience

In relation to user experience, we should focus on the connection between web interface and our search engine system in the first place, this is to ensure the compatibility of related classes, databases, and interface adaptations.

Adding features like autosuggestions and spell-checking would enrich the user experience and make our search engine more user centric.

Enhanced Query processing/Ranking

On the other hand, we could embed synonym recognition methods as additional features to enhance our search engine's ability to understand user intent and deliver more accurate search results. We also could design the system with scalability in mind to handle large-scale data volumes and user queries.

To sum up, our search engine system demonstrates strengths in efficient crawling, indexing, and ranking. However, there are areas for improvement, including enhancing the query processing, incorporation advanced ranking techniques, enriching the user experience, and optimizing scalability and performance. We believe that by addressing these aspects, our search engine can evolve into a more powerful and user-friendly solution.

Contribution

Names	Yuen Man Him	Tsang Hing Ki	Chan Char-ju Julius
Contribution %	35%	50%	15%