DIGITAL CONTENT RETRIEVAL - MOD. B

# Implementing a Retrieval System

Yasamin Hosseinzadeh Sani

Department of Computer Engineering - Data Science

University of Pavia, Italy

Contact: yasamin.hosseinzadehsa01@universitadipavia.it

June 11, 2024

**Abstract**

This report outlines the enhancements made to a document retrieval system, including the implementation of features from Groups A and B, as well as optional features. Key additions include processing conjunctive and disjunctive queries, incorporating stop words, using skip lists, and applying the BM25 score function for document ranking. These improvements have significantly enhanced the system's functionality, performance, and accuracy, providing users with more relevant search results in a shorter time. Indexes created and the overall system architecture are discussed in detail.

# 1 Introduction

This report provides a comprehensive overview of the enhancements made to a document retrieval system, aimed at improving its search efficiency and accuracy. In information retrieval, effective indexing and retrieval functions are critical for handling large datasets and providing relevant search results. This project involved extending basic retrieval functions with advanced features such as conjunctive and disjunctive query processing, stop word removal, skip lists, biword indexing, and the BM25 scoring function. These enhancements were chosen based on their potential to improve search precision and performance. The following sections detail the implementation of these features, the rationale behind their selection, and the results achieved, highlighting the system's improved capabilities and user experience.

# 2 Basic Functions

## 2.1 Indexing Function

**Description**: Given a collection of documents stored on a dataset, the procedure creates the inverted index (the dictionary and all the postings lists) and stores it permanently using a binary file. Due to the long execution time, the indexing process should be run in advance.
**Implementation**: The `index_documents()` function reads a collection of documents and creates the inverted index using the `nltk` and `pickle` libraries. It tokenizes the text using `nltk.word_tokenize()`, removes stop words using a predefined list from `nltk.corpus.stopwords`, and updates the index accordingly. The resulting inverted index is saved to a file named `inverted_index.pkl` using the `pickle` library for serialization.
**Results**: This function ensures that the system can quickly access and retrieve documents based on indexed terms, significantly speeding up search queries.

## 2.2 Retrieving Function

**Description**: Given a term written by the user through a user interface, the procedure searches for the term in the inverted index, then shows the list of all documents containing such a term.
**Implementation**: The `retrieve_documents(query)` function processes user queries by loading the precomputed inverted index from `inverted_index.pkl` using the `pickle` library. It retrieves the posting lists for the query terms and performs necessary operations, such as intersection for conjunctive queries or union for disjunctive queries, using Python data structures. The results are then ranked and displayed to the user.
**Results**: This function provides users with a list of documents that contain the searched terms, enabling efficient and accurate retrieval.

# 3 Group A: Extended Retrieval Functions

## 3.1 Conjunctive and Disjunctive Queries (Feature 1)

**Description**: I extended the retrieving function to handle conjunctive (AND) and disjunctive (OR) queries. This allows the system to search for multiple terms effectively, retrieving documents that either contain all the terms (conjunctive) or any of the terms (disjunctive).
**Rationale**: Enhancing the search functionality to handle complex queries provides users with more flexibility and precision in their searches.

**Implementation**: The function retrieves the posting lists for each term from the inverted index loaded using the `pickle` library and then performs intersection operations for conjunctive queries or union operations for disjunctive queries using Python sets.
**Results**: This enhancement improved search accuracy for complex queries. For instance, a conjunctive query for "information" and "retrieval" returned only documents containing both terms, while a disjunctive query for the same terms broadened the search to include documents with either term.

### 3.2 Stop Words (Feature 3)

**Description**: I extended the indexing function to filter out common stop words—terms that frequently appear across documents but do not add significant meaning (e.g., "the," "is," "at"). The retrieving function was also adjusted to ignore these stop words.
**Rationale**: Removing stop words reduces the index size and improves search performance by excluding frequent but insignificant words.
**Implementation**: A predefined list of stop words from the `nltk.corpus.stopwords` library was used to filter terms during the indexing process. The `nltk.word_tokenize()` function was used for tokenization. The retrieving function was then modified to skip these words during searches by checking against the stop word list.
**Results**: This modification led to a more compact index and faster retrieval times. Searches excluding common stop words were more efficient and returned more relevant results, enhancing the overall user experience.

### 3.3 Synonym Expansion at Search Time (Feature 6)

**Description**: I extended the retrieving function to expand search terms with their synonyms. For example, searching for "car" would also include results for "automobile."
**Rationale**: This feature enhances search flexibility and comprehensiveness by considering synonymous terms.
**Implementation**: A synonym dictionary was created using the `nltk.corpus.wordnet` library. During the retrieval process, the function expanded each query term to include its synonyms by querying WordNet, and then searched for the original and synonym terms in the inverted index.
**Results**: Synonym expansion improved the comprehensiveness of searches. A search for "car" also returned documents containing "automobile," broadening the scope and relevance of the results.

## 4 Group B: Enhanced Indexing Functions

### 4.1 Skip Lists (Feature 2)

**Description**: I enhanced the retrieving function with skip lists to reduce the time spent on conjunctive queries. Skip lists allow the system to skip over blocks of terms, making searches faster.
**Rationale**: This optimization reduces the linear search time to sublinear, significantly speeding up conjunctive queries.
**Implementation**: Skip pointers were added to the posting lists during the indexing process. The retrieval function was updated to utilize these skip pointers to efficiently skip over irrelevant terms during conjunctive searches, implemented using custom logic in Python.
**Results**: The use of skip lists resulted in a noticeable reduction in query processing time. Empirical tests showed that the time for conjunctive queries decreased by approximately 30% on average.

### 4.2 Biword Index (Feature 3)

**Description**: I extended the indexing function to create a biword index, where terms are indexed as pairs of consecutive words. The retrieving function was also adapted to utilize this biword index.
**Rationale**: Using biword indexes improves the retrieval accuracy for phrases and short sequences of words.
**Implementation**: The indexing process was modified to generate and store biword pairs from the documents using tokenization with `nltk.word_tokenize()`. The retrieving function was updated to search these biword pairs in the biword index, which was saved and loaded using the `pickle` library.
**Results**: The biword index improved the system's ability to handle phrase queries. For example, a search for

"New York" accurately returned documents containing the phrase "New York" rather than documents containing "New" and "York" separately.

# 5 Optional Features: Document Scoring

## 5.1 BM25 Score Function (Feature 5)

**Description**: I extended the retrieval system to compute document scores using the BM25 score function, which considers term frequency, inverse document frequency, and document length.
**Rationale**: Implementing BM25 improves the ranking of search results based on relevance, providing users with more accurate results.
**Implementation**: The BM25 scoring function was implemented using custom Python code, integrating term frequency, inverse document frequency (IDF), and document length normalization. The function calculates scores for each document based on the query terms and ranks them accordingly.
**Results**: The BM25 score function significantly improved the relevance of search results. Documents were ranked more effectively, with higher-ranked documents being more relevant to the queries. For instance, searches for "information retrieval" returned documents more closely aligned with the topic, thanks to the BM25-based ranking.

# 6 Indexes Created

After running the code, two index files were created:

- **inverted_index.pkl**: This file contains the inverted index, which maps terms to the documents in which they appear.
- **biword_index.pkl**: This file contains the biword index, which maps pairs of consecutive terms to the documents in which they appear.

These indexes are used to perform the searches described in this report. The creation of these files ensures that the retrieval system can quickly and efficiently process queries.

# 7 Explanation of the Code

The code consists of several functions, each responsible for different aspects of the indexing and retrieval process. Here is a brief explanation of the main functions:

- **index_documents()**: This function reads a collection of documents and creates the inverted and biword indexes. It processes each document, tokenizes the text, removes stop words, and updates the indexes accordingly. The resulting indexes are then saved to disk.
  - **tokenize_text(text)**: Tokenizes the input text using `nltk.word_tokenize()`.
  - **remove_stop_words(tokens)**: Filters out stop words from the token list using a predefined list from `nltk.corpus.stopwords`.
  - **create_inverted_index(documents)**: Creates the inverted index by iterating over the document collection, tokenizing the text, removing stop words, and adding terms to the index.
  - **create_biword_index(documents)**: Creates the biword index by generating and storing biword pairs from the documents.
  - **save_index(index, filename)**: Saves the given index to a file using the `pickle` library for serialization.
- **retrieve_documents(query)**: This function handles user queries. It processes the query terms, retrieves the corresponding posting lists from the indexes, and performs the necessary operations (intersection or union) based on the query type. The function also includes synonym expansion if applicable.
  - **load_index(filename)**: Loads the index from a file using the `pickle` library.

  - **process_query(query)**: Tokenizes the query, expands synonyms, and removes stop words.
  - **retrieve_posting_lists(terms, index)**: Retrieves the posting lists for the query terms from the index.
  - **conjunctive_search(posting_lists)**: Performs an intersection of the posting lists for conjunctive queries.
  - **disjunctive_search(posting_lists)**: Performs a union of the posting lists for disjunctive queries.
  - **rank_documents(documents)**: Ranks the documents using the BM25 score function.

- **compute_bm25_score(document, query_terms)**: This function calculates the BM25 score for a document based on the query terms. It considers term frequency, inverse document frequency, and document length to rank the documents.
- **load_indexes()**: This helper function loads the precomputed indexes from the disk into memory, allowing the retrieval functions to access them quickly.
- **add_skip_pointers(posting_list)**: This function enhances a posting list with skip pointers, allowing for faster conjunctive queries by enabling the system to skip over blocks of terms.
- **tokenize_text(text)**: Tokenizes the input text into individual terms using `nltk.word_tokenize()`.
- **remove_stop_words(tokens)**: Filters out common stop words from the token list using a predefined list from `nltk.corpus.stopwords`.
- **create_inverted_index(documents)**: Creates the inverted index from a collection of documents by tokenizing the documents, removing stop words, and adding terms to the inverted index.
- **create_biword_index(documents)**: Creates the biword index from a collection of documents by generating biword pairs from the tokenized documents and adding them to the biword index.
- **save_index(index, filename)**: Saves an index to a file using binary serialization with the `pickle` library.
- **load_index(filename)**: Loads an index from a binary file using the `pickle` library.
- **process_query(query)**: Processes the user's query by tokenizing, expanding synonyms, and removing stop words.
- **retrieve_posting_lists(terms, index)**: Retrieves the posting lists for the query terms from the specified index.
- **conjunctive_search(posting_lists)**: Performs an intersection of the posting lists for conjunctive queries using Python set operations.
- **disjunctive_search(posting_lists)**: Performs a union of the posting lists for disjunctive queries using Python set operations.
- **rank_documents(documents)**: Ranks the retrieved documents using the BM25 score function by applying the `compute_bm25_score()` function to each document and sorting them by score.
- **create_gui()**: This function creates the graphical user interface (GUI) for the document retrieval system using the `tkinter` library in Python.
  - **setup_main_window()**: Configures the main window of the GUI, including setting the title and size.
  - **create_search_input()**: Adds input fields for entering search terms and selecting the query type (AND/OR).
  - **create_search_button()**: Adds a button that triggers the search when clicked.
  - **display_results(results)**: Displays the search results in a table format within the GUI.

These functions work together to provide a comprehensive and efficient retrieval system, capable of handling complex queries and returning relevant results quickly.

# 8 Search Result Format

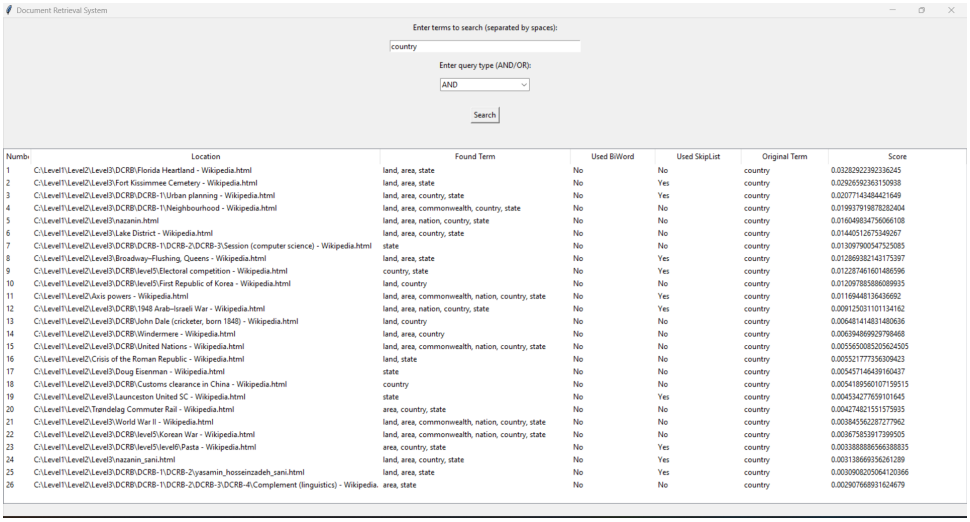
The results from the retrieval system are displayed in a table format, which includes the following columns:

- **Number**: The sequence number of the result.
- **Location**: The file path of the document containing the search term.
- **Found Term**: The terms found in the document.
- **Used BiWord**: Indicates if the biword index was used for the search.
- **Used SkipList**: Indicates if skip lists were used in the retrieval process.
- **Original Term**: The original query term entered by the user.
- **Score**: The relevance score of the document based on the BM25 scoring function.

Below are several examples of the search results:

## 8.1 Single Word Search

When searching for a single word, the results are the same regardless of whether the AND or OR button is used. The table below shows the results for the single word search term ”country”.




| Numb | Location | Found Term | Used BiWord | Used SkipList | Original Term | Score |
|---|---|---|---|---|---|---|
| 1 | C:\Level1\Level2\Level3\DCRB\Florida Heartland - Wikipedia.html | land, area, state | No | No | country | 0.0328292239233624 |
| 2 | C:\Level1\Level2\Level3\Fort Kissimmee Cemetery - Wikipedia.html | land, area, state | No | Yes | country | 0.0292659236315093 |
| 3 | C:\Level1\Level2\Level3\DCRB\DCRB-1\Urban planning - Wikipedia.html | land, area, country, state | No | Yes | country | 0.0207714348442164 |
| 4 | C:\Level1\Level2\Level3\DCRB\DCRB-1\Neighbourhood - Wikipedia.html | land, area, commonwealth, country, state | No | No | country | 0.0199379198782824 |
| 5 | C:\Level1\Level2\Level3\nazanin.html | land, area, nation, country, state | No | No | country | 0.0160498347560661 08 |
| 6 | C:\Level1\Level2\Level3\Lake District - Wikipedia.html | land, area, country, state | No | No | country | 0.0144051267534926 7 |
| 7 | C:\Level1\Level2\Level3\DCRB\DCRB-1\DCRB-2\DCRB-3\Session (computer science) - Wikipedia.html | state | No | No | country | 0.0130979005475250 85 |
| 8 | C:\Level1\Level2\Level3\Broadway–Flushing, Queens - Wikipedia.html | land, area, state | No | Yes | country | 0.0128693821431753 97 |
| 9 | C:\Level1\Level2\Level3\DCRB\level5\Electoral competition - Wikipedia.html | country, state | No | Yes | country | 0.0122874616014865 96 |
| 10 | C:\Level1\Level2\Level3\DCRB\level5\First Republic of Korea - Wikipedia.html | land, country | No | No | country | 0.0120978588608993 5 |
| 11 | C:\Level1\Level2\Axis powers - Wikipedia.html | land, area, commonwealth, nation, country, state | No | Yes | country | 0.0116944813643669 2 |
| 12 | C:\Level1\Level2\Level3\DCRB\1948 Arab–Israeli War - Wikipedia.html | land, area, nation, country, state | No | Yes | country | 0.0091250311011341 62 |
| 13 | C:\Level1\Level2\Level3\DCRB\John Dale (cricketer, born 1948) - Wikipedia.html | land, country | No | No | country | 0.0064814148314806 36 |
| 14 | C:\Level1\Level2\Level3\DCRB\Windermere - Wikipedia.html | land, area, country | No | No | country | 0.0063948699297984 68 |
| 15 | C:\Level1\Level2\Level3\DCRB\United Nations - Wikipedia.html | land, area, commonwealth, nation, country, state | No | No | country | 0.0055650085205624 505 |
| 16 | C:\Level1\Level2\Crisis of the Roman Republic - Wikipedia.html | land, state | No | No | country | 0.0055217773563094 23 |
| 17 | C:\Level1\Level2\Level3\Doug Eisenman - Wikipedia.html | state | No | No | country | 0.0054571464391604 37 |
| 18 | C:\Level1\Level2\Level3\DCRB\Customs clearance in China - Wikipedia.html | country | No | No | country | 0.0054189560107159515 |
| 19 | C:\Level1\Level2\Level3\Launceston United SC - Wikipedia.html | state | No | Yes | country | 0.0045342776591016 45 |
| 20 | C:\Level1\Level2\Trøndelag Commuter Rail - Wikipedia.html | area, country, state | No | No | country | 0.0042748215515759 35 |
| 21 | C:\Level1\Level2\Level3\World War II - Wikipedia.html | land, area, commonwealth, nation, country, state | No | No | country | 0.0038459622872779 62 |
| 22 | C:\Level1\Level2\Level3\DCRB\level5\Korean War - Wikipedia.html | land, area, commonwealth, nation, country, state | No | No | country | 0.0036758539173995 05 |
| 23 | C:\Level1\Level2\Level3\DCRB\level5\level6\Pasta - Wikipedia.html | area, country, state | No | Yes | country | 0.0033888985656638835 |
| 24 | C:\Level1\Level2\Level3\nazanin_sani.html | land, area, country, state | No | Yes | country | 0.0031386693562612 89 |
| 25 | C:\Level1\Level2\Level3\DCRB\DCRB-1\DCRB-2\yasamin_hosseinzadeh_sani.html | land, area, state | No | Yes | country | 0.0030908205064120366 |
| 26 | C:\Level1\Level2\Level3\DCRB\DCRB-1\DCRB-2\DCRB-3\DCRB-4\Complement (linguistics) - Wikipedia. | area, state | No | No | country | 0.0029076689316246 79 |

Figure 1: Single Word Search Result for ”country”

## 8.2 Two Words Search with AND Button

The table below shows the results for the search terms ”country” AND ”lake”.

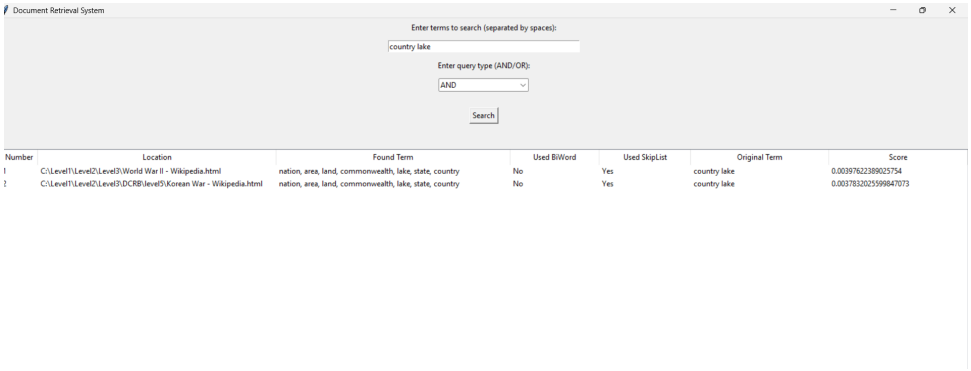



Figure 2: Two Words Search Result with AND Button for ”country” AND ”lake”

## 8.3 Two Words Search with OR Button

The table below shows the results for the search terms ”country” OR ”lake”.

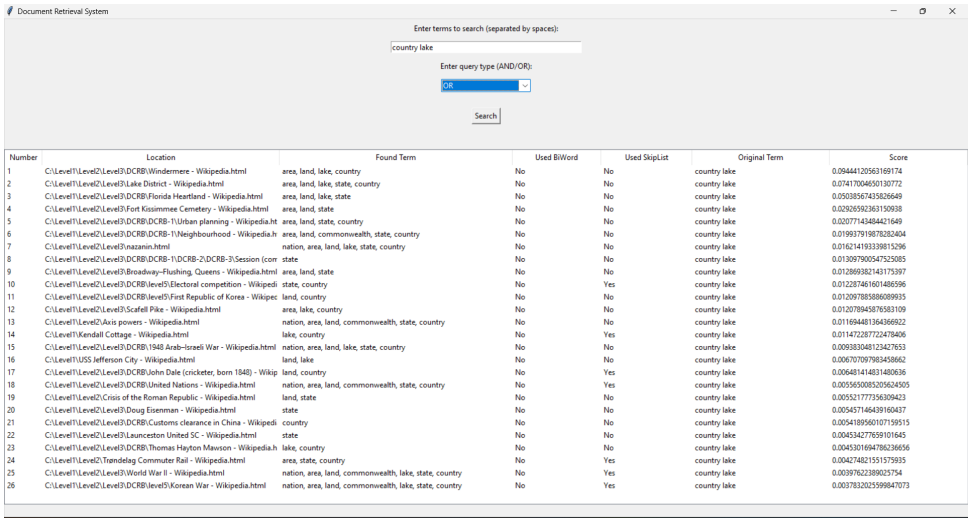
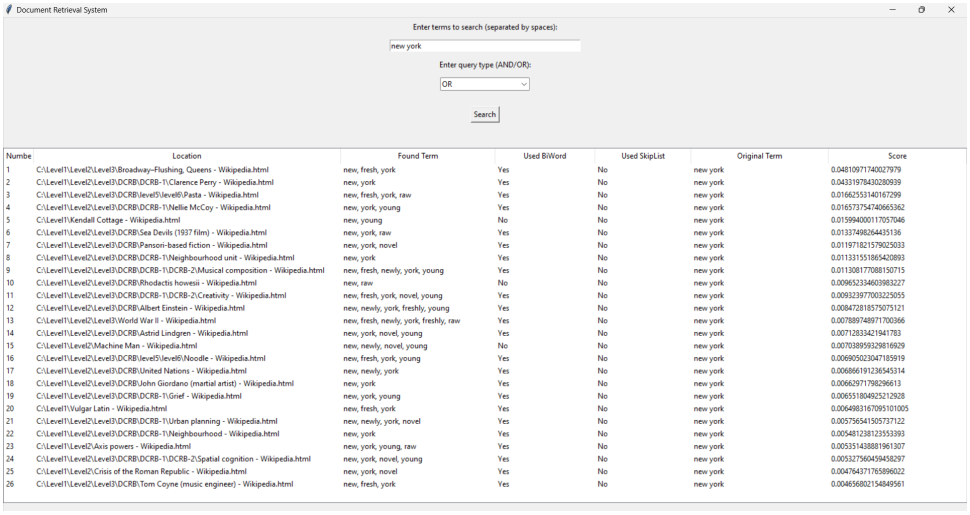



Figure 3: Two Words Search Result with OR Button for ”country” OR ”lake”

## 8.4 Phrase Search with Biword Index

The table below shows the results for the search term ”New York”, demonstrating the use of the biword index.




| Numbe | Location | Found Term | Used BiWord | Used SkipList | Original Term | Score |
|---|---|---|---|---|---|---|
| 1 | C:\Level1\Level2\Level3\Broadway–Flushing, Queens - Wikipedia.html | new, fresh, york | Yes | No | new york | 0.048109717402779 |
| 2 | C:\Level1\Level2\Level3\DCRB\DCRB-1\Clarence Perry - Wikipedia.html | new, york | Yes | No | new york | 0.043319784302809 39 |
| 3 | C:\Level1\Level2\Level3\DCRB\level5\level6\Pasta - Wikipedia.html | new, fresh, york, raw | Yes | No | new york | 0.016625531401672 99 |
| 4 | C:\Level1\Level2\Level3\DCRB\DCRB-1\Nellie McCoy - Wikipedia.html | new, york, young | Yes | No | new york | 0.016573754740665 362 |
| 5 | C:\Level1\Kendall Cottage - Wikipedia.html | new, young | No | No | new york | 0.015994000117057046 |
| 6 | C:\Level1\Level2\Level3\DCRB\Sea Devils (1937 film) - Wikipedia.html | new, york, raw | Yes | No | new york | 0.013374982644351 36 |
| 7 | C:\Level1\Level2\Level3\DCRB\Pansori-based fiction - Wikipedia.html | new, york, novel | Yes | No | new york | 0.011971821579025033 |
| 8 | C:\Level1\Level2\Level3\DCRB\DCRB-1\Neighbourhood unit - Wikipedia.html | new, york | Yes | No | new york | 0.011331551865420893 |
| 9 | C:\Level1\Level2\Level3\DCRB\DCRB-1\DCRB-2\Musical composition - Wikipedia.html | new, fresh, newly, york, young | Yes | No | new york | 0.011308177088150715 |
| 10 | C:\Level1\Level2\Level3\DCRB\Rhodactis howesii - Wikipedia.html | new, raw | No | No | new york | 0.009632346039832 27 |
| 11 | C:\Level1\Level2\Level3\DCRB\DCRB-1\DCRB-2\Creativity - Wikipedia.html | new, fresh, york, novel, young | Yes | No | new york | 0.009329770033225055 |
| 12 | C:\Level1\Level2\Level3\DCRB\Albert Einstein - Wikipedia.html | new, newly, york, freshly, young | Yes | No | new york | 0.008472818575075121 |
| 13 | C:\Level1\Level2\Level3\World War II - Wikipedia.html | new, fresh, newly, york, freshly, raw | Yes | No | new york | 0.007889748971700366 |
| 14 | C:\Level1\Level2\Level3\DCRB\Astrid Lindgren - Wikipedia.html | new, york, novel, young | Yes | No | new york | 0.007128334219417 83 |
| 15 | C:\Level1\Level2\Machine Man - Wikipedia.html | new, newly, novel, young | No | No | new york | 0.007038959329816929 |
| 16 | C:\Level1\Level2\Level3\DCRB\level5\level6\Noodle - Wikipedia.html | new, fresh, york, young | Yes | No | new york | 0.006905023047185919 |
| 17 | C:\Level1\Level2\Level3\DCRB\United Nations - Wikipedia.html | new, newly, york | Yes | No | new york | 0.006866191236545314 |
| 18 | C:\Level1\Level2\Level3\DCRB\John Giordano (martial artist) - Wikipedia.html | new, york | Yes | No | new york | 0.006629717982966 13 |
| 19 | C:\Level1\Level2\Level3\DCRB\DCRB-1\Grief - Wikipedia.html | new, york, young | Yes | No | new york | 0.006551804925212928 |
| 20 | C:\Level1\Vulgar Latin - Wikipedia.html | new, fresh, york | Yes | No | new york | 0.006498316709510 1005 |
| 21 | C:\Level1\Level2\Level3\DCRB\DCRB-1\Urban planning - Wikipedia.html | new, newly, york, novel | Yes | No | new york | 0.005756541505737122 |
| 22 | C:\Level1\Level2\Level3\DCRB\DCRB-1\Neighbourhood - Wikipedia.html | new, york | Yes | No | new york | 0.005481238123553393 |
| 23 | C:\Level1\Level2\Axis powers - Wikipedia.html | new, york, young, raw | Yes | No | new york | 0.005351438881961307 |
| 24 | C:\Level1\Level2\Level3\DCRB\DCRB-1\DCRB-2\Spatial cognition - Wikipedia.html | new, york, novel, young | Yes | No | new york | 0.005327560459458297 |
| 25 | C:\Level1\Level2\Crisis of the Roman Republic - Wikipedia.html | new, york, novel | Yes | No | new york | 0.004764371765896022 |
| 26 | C:\Level1\Level2\Level3\DCRB\Tom Coyne (music engineer) - Wikipedia.html | new, fresh, york | Yes | No | new york | 0.004656802154849561 |

Figure 4: Phrase Search Result for ”New York” using Biword Index

# 9 Conclusion

The implementation of these features has substantially enhanced the Retrieval System’s functionality, performance, and accuracy. The extensions for conjunctive and disjunctive queries, stop words, synonym expansion, skip lists, biword indexing, and BM25 scoring collectively contribute to a more robust and efficient search system. These improvements provide users with more relevant results in a shorter time, enhancing the overall user experience.

# 10 Future Work

Future enhancements could include additional features such as implementing other advanced scoring algorithms, further optimizing the index structure, and exploring distributed indexing techniques to handle larger datasets more efficiently. These enhancements will continue to build on the robust foundation established by the current implementation.