Information Retrieval System

Inverted Index and Retrieval Functions

Raffaele Cammi

Master's Degree in Computer Science Multimedia
University of Pavia

Exam: Information Retrieval

Project Overview

- Implementation of a basic Information Retrieval (IR) system.
- Two core functions:
  - **Indexing**: build and store an inverted index from a document collection.
  - **Retrieving**: given a query term, return all documents containing that term.
- At least one extension from **Group A** and one from **Group B**.
- This project focuses on: Stop Words (A), Porter Stemming (B), and Skip Lists (B).

System Architecture

Main Components

- `Tokenizer` — splits text into terms.
- `StopWords` — removes frequent or irrelevant tokens.
- `PorterStemmer` — reduces words to root form.
- `Indexer` — builds dictionary and posting lists.
- `IndexIO` — persists and loads the index.
- `Retriever` — processes queries and returns results.
- `GUI` — user interface for searches.

Data Flow

- Documents → Tokenizer → StopWords → PorterStemmer → Indexer → IndexIO
- Inverted Index → Retriever → GUI

Indexing Function

Goal: Build the inverted index from a text collection.

1. Tokenize documents into words.
2. Remove stop words.
3. Apply Porter stemming.
4. Update dictionary and posting lists.
5. Persist the index to disk via `IndexIO`.

Output Files

- `index.dict` — dictionary and posting lists
- `docs.map` — document ID $\leftrightarrow$ path
- `collection.freq` — global term frequencies

Example entry: `term|df|docID:tf,docID:tf,...`

Data Structures

Dictionary

- `HashMap<String, PostingList>` mapping `term` $\rightarrow$ `PostingList`.

PostingList

- `ArrayList<Posting>` storing occurrences of a term across documents.
- Each `Posting` = (`docID`, `term frequency`, optional `skip pointer`).

Document Map

- `Map<Integer, String>` mapping internal `docID` to file path.

Skip Pointers

- Added to posting lists to accelerate intersections in conjunctive queries.

Retrieving Function

Goal: Retrieve documents matching the user query.

1. Load index from disk (`IndexIO`).
2. Normalize query (tokenize, remove stop words, stem).
3. Lookup term(s) in the dictionary.
4. Retrieve posting lists and intersect/union as needed.
5. Return matching document IDs and paths (shown in GUI).

Supported Features

- Single-term and multi-term queries.
- Conjunctive (AND) and disjunctive (OR) modes.
- StopWords filtering and Porter stemming.
- Skip pointers for faster AND intersections.

Implemented Extensions

Group	Feature	Description
A	Stop Words	Removes frequent, low-information terms
B	Porter Stemmer	Normalizes morphological variants
B	Skip List	Sublinear conjunctive intersections

Benefits

- Cleaner index, fewer redundant terms.
- Better recall for morphological variations.
- Faster AND queries with skip optimization.

Persistence and File Formats

Files generated by `IndexIO`

1. `index.dict`: dictionary + posting lists.
2. `docs.map`: mapping from internal `docID` to file path.
3. `collection.freq`: collection frequency for each term.

Advantages

- Reload index without re-indexing.
- Human-readable, text-based format.
- Easier inspection and debugging.

GUI and Execution

GUI Features

- Build or load the index.
- Enter single or multi-term queries (AND / OR).
- Display matching documents (IDs and paths).

Demo

Conclusions

Achievements

- Functional inverted index and retrieval pipeline.
- Integrated StopWords, Porter Stemming, and Skip Lists.
- Persistent, efficient on-disk structure.