Name: Hetu Patel - 501215707
Course: CPS842
Assignment 1: Inverted Index

1. Overview

The goal of this assignment was to build an inverted index from the CACM research paper collection and create a small search tool. My invert.py program reads the CACM dataset, tokenizes the text, removes optional stop words, stems terms using a light version of Porter's algorithm, and outputs two files: a dictionary and a postings list. My test.pyprogram then loads those files so I can type a term and see which documents contain it, plus frequency, positions, and a short snippet.

2. Parsing & Tokenization

I wrote a parser to extract the required fields (.I, .T, .W, .B, .A). Only the **title and abstract** are used for indexing, as instructed. The text is lowercased and split into words using a simple regular expression. I can turn stop word removal on or off using the flag --stopwords_on. The stop word list I used comes from the provided stopwords.txt.

For stemming, I implemented a compact version of Porter's algorithm. It reduces words like *computing* to *comput* so related terms match.

3. Data Structures

I used a Python dictionary (dict) for fast insertion and lookup while processing each document:

index = {term: {doc_id: [positions...]}}

- Each term maps to another dictionary of documents where it appears.
- Each document stores a list of all token positions for that term.

After processing all documents, I:

- Sort the terms alphabetically (required).
- Sort the document IDs for each term.
- Write the dictionary and postings so that **line N in dictionary.txt matches line N in postings.txt.**

4. Output Format

- **dictionary.txt** — term<TAB>doc_freq
- **postings.txt** — JSON lines:

{"term": "algorithm", "postings": [{"doc_id": 12, "tf": 3, "positions": [4, 17, 90]}]}

This one-to-one mapping was important to maintain.

5. Testing Program

The test.py program:

- Loads the dictionary and postings into memory.
- Lets me type a single term and prints:
    - Document frequency (DF)
    - Each matching doc's ID, title, term frequency, and positions
    - A snippet of about 10 words around the first match with the term highlighted
- Measures and prints the search response time in milliseconds for each query and the average when I exit with ZZEND.

6. How to Run

1. Build the index:

    python3 invert.py --input ./cacm.tar.gz --stopwords ./stopwords.txt --output-dir ./out --stemming on --stopwords_on on

2. Query terms:

    python3 test.py --index-dir ./out

3. Type terms like algorithm or comput; type ZZEND to quit.

7. Reflections / Design Choices

- Using Python's hash map made inserts O(1) and kept the build fast.
- Sorting at the end ensures we meet the alphabetical and doc_id ordering requirement.
- Outputting JSON postings made debugging easy and transparent.
- The snippet feature helps test correctness because I can see the context around the term.

8. Limitations & Next Steps

- Tokenizer is simple and doesn't handle all edge cases (e.g., hyphenated terms).
- Stemming is a basic Porter implementation; a full library could be more accurate.
- For large collections, compression (like gap encoding) could reduce index size.

9. Appendix

Screenshots of the codes and results