


CS 172: Group 29 Project – Part B Report

Title: Search Engine Indexing and Web Interface for .edu Research Pages

Video Demo Link :  group29-demovideo.mp4

1. Collaboration Details

- **Chandana Anand Rangappa (862545654):**
Developed the PyLucene-based indexer (edu_indexer.py) and built the full-stack Flask interface (app.py). Implemented query parsing, search handling, and integrated the snippet generation logic. Conducted end-to-end testing of the system.
- **Ranjitha Narasimhamurthy (862548883):**
Wrote the HTML parser using BeautifulSoup to extract titles and bodies from .html files. Applied filtering logic and ensured indexable quality of content.
- **Akshit Sharma (862549032):**
Created the custom snippet generation logic that extracts and displays meaningful excerpts from matched documents.
- **Hrutvika Mutteppwar (862546800):**
Authored the Unix/Linux indexer.sh script to automate indexing. Also conducted indexing performance evaluations.
- **Vismaya Anand Bolbandi (862548529):**
Assisted in frontend integration and evaluation testing. Captured screenshots and recorded the video demo for final submission.

2. Overview of the System

a. Architecture

The project contains three major components:

1. **HTML Content Parser** – Extracts title and body from crawled pages located in ./data/html_pages/.
2. **Indexer (edu_indexer.py)** – Tokenizes and indexes HTML content using PyLucene.
3. **Web App (app.py)** – Runs a Flask-based web server that enables querying the Lucene index.

Data Flow:

[HTML Files] → edu_indexer.py → Lucene Index → app.py → Browser UI

b. Index Structures

- **Title Field:** Boosted (2.0x), indexed and stored.
- **Body Field:** Indexed using StandardAnalyzer.
- **Path/URL Field:** Stored for back-reference but not indexed.

c. Search Algorithm

- **Ranking Method:** BM25 relevance scoring.
- **Query Parsing:** Supports keyword search on both title and body using QueryParser.
- **Snippet Generation (Extra Credit):**
 - Manually extracts the sentence containing the first query keyword.
 - Provides meaningful preview not generated by Lucene's default mechanism.

3. Limitations of the System

- Indexes must be manually rebuilt if new pages are added.
- Advanced search features like phrase queries and filters are not implemented.
- Snippets do not currently highlight matched keywords.
- No spell correction or synonym expansion is included.

4. Instructions on How to Deploy the System

A. Indexing

Ensure your system has Python 3.10+ and PyLucene installed. Then run:

```
[user@server] ./indexer.sh ./data/html_pages
```

This will process the HTML files and build an index at ./index/.

B. Launching the Search Interface

Start the Flask server:

```
[user@server] python3 app.py
```

Navigate to:

<http://127.0.0.1:5000>

Included Files:

- edu_indexer.py – HTML indexing script
- app.py – Flask web server for querying
- templates/index.html – Frontend UI template
- indexer.sh – Bash script to run the indexer
- requirements.txt – Dependency list

5. Web Application Overview

Key features:

- **Search Textbox:** Accepts keyword-based user queries.
- **Search Button:** Executes a ranked search.
- **Result Display:** Top 10 results ordered by BM25 score.

Each result includes:

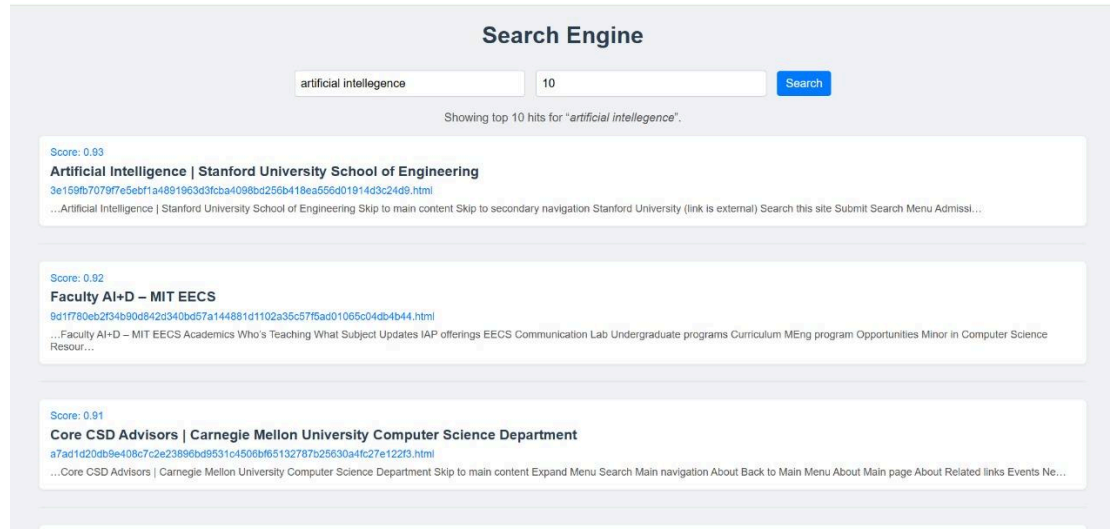
- Title from the indexed document
- Path or reference to the source HTML file
- **Custom snippet** matching the query term context

6. Screenshots of the System in Action

IndexerOutput

```
usage: python3 edu_indexer.py -m name_dir -i index_dir
cs172@class-069:~/cs172_project$ python3 edu_indexer.py ./data/html_pages ./edu_lucene_index
Indexed 100 documents so far...
Indexed 200 documents so far...
Indexed 300 documents so far...
Indexed 400 documents so far...
Indexed 500 documents so far...
Indexed 600 documents so far...
Indexed 700 documents so far...
Indexed 800 documents so far...
Indexed 900 documents so far...
Indexed 1000 documents so far...
```

Search Interface and Result Snippet Example



7. Conclusion

In this project, we extended our focused academic crawler from Part A into a complete, searchable information retrieval system. By leveraging PyLucene for indexing and Flask for the web interface, we successfully implemented a lightweight academic search engine tailored to .edu research pages. The system demonstrates key principles of information retrieval, including tokenization, inverted indexing, BM25 ranking, and result presentation. Additionally, we implemented a custom snippet generation feature that enhances user experience by displaying contextually relevant text for each result — fulfilling the extra credit requirement. While the system currently supports basic keyword search over crawled HTML pages, it lays a solid foundation for future enhancements such as dynamic indexing, advanced query features, and semantic search. Overall, the project showcases how classical IR techniques can be applied in a modern web environment to deliver meaningful and relevant search experiences over academic content.