CS-172: Introduction to Information Retrieval
Title: Web Crawler for .edu Domain Research Pages

Collaboration Details

- Ranjitha Narasimhamurthy (862548883):: Implemented URL normalization and SHA-256 hashing for duplicate prevention.
- Chandana Anand Rangappa (862545654): Designed the BFS controller using deque and depth/page limiting logic.
- Akshit Sharma (862549032): Developed link extraction and domain filtering for .edu-only pages.
- Hrutvika Muttepwar (862546800): Managed HTML file writing using hashed filenames.
- Vismaya Anand Bolbandi (862548529): Created crawler.bat for Windows execution and handled integration/testing.

Abstract

This project presents a focused web crawler that collects academic research pages from university websites under the .edu domain. It employs a breadth-first search (BFS) strategy and traverses hyperlinks up to a specified depth. The crawler filters for HTML pages, stores them in a raw format using unique hashed filenames, and stops once a maximum page count is reached. This tool lays the foundation for future indexing and search in Part B of the course project.

Introduction

Web crawlers are essential for building search engines and data pipelines. Academic websites under the .edu domain are rich in research content, faculty listings, publications, and labs. This project implements a web crawler in Python to collect HTML content starting from research-focused seed pages. The crawler supports depth-based traversal, domain filtering, duplicate prevention, and structured HTML saving, meeting all Part A requirements of the CS172 course project.

System Overview

Architecture

1. **URL Queue System**
    - Implements BFS using collections.deque.
    - Each URL is normalized using urllib.parse to ensure consistency (removes query strings, fragments).
    - SHA-256 is used to hash each normalized URL for visited tracking.
2. **Crawler Controller**
    - Accepts seed.txt, max depth, and max pages as parameters.

- Processes URLs until either the page limit is hit or the queue is exhausted.
- Skips already visited URLs by checking their hash.

3. **Page Fetching**
    - Uses requests to make HTTP GET calls with a custom user-agent.
    - Skips non-HTML pages (e.g., PDFs) based on the Content-Type header.
    - Errors like 403 or timeouts are caught and printed, not crashing the crawl.
4. **Link Extraction & Filtering**
    - Extracts <a href> links using BeautifulSoup.
    - Uses urljoin to convert relative links to absolute URLs.
    - Filters only links that contain ".edu" in their domain using urlparse.
5. **HTML Saving**
    - Each valid HTML page is saved as a raw .html file using its SHA-256 hash as the filename.
    - Files are stored in the ./data/html_pages directory.

Crawling Strategy

- Breadth-First Search: URLs are processed in FIFO order, ensuring all pages at depth d are visited before depth d+1.
- Domain Filtering: Only .edu domain links are allowed to continue the crawl.
- Depth-Limited: If the link's depth exceeds the user-defined max depth, it is skipped.
- Duplicate Filtering: Each URL is normalized and hashed; only unseen hashes are processed.
- Page Limit: Crawling stops when the number of HTML pages reaches max_pages.

Screenshot 1: Crawler Output in CMD

This screenshot shows the execution of the command:

```
crawler.bat seed1.txt 4 50000
```

The crawler began crawling from multiple .edu seed URLs and used a breadth-first traversal strategy. It logs each page being visited, along with HTTP status codes. Pages that returned 404 errors were gracefully skipped. This demonstrates the crawler's real-time feedback, depth tracking, and error handling mechanisms.

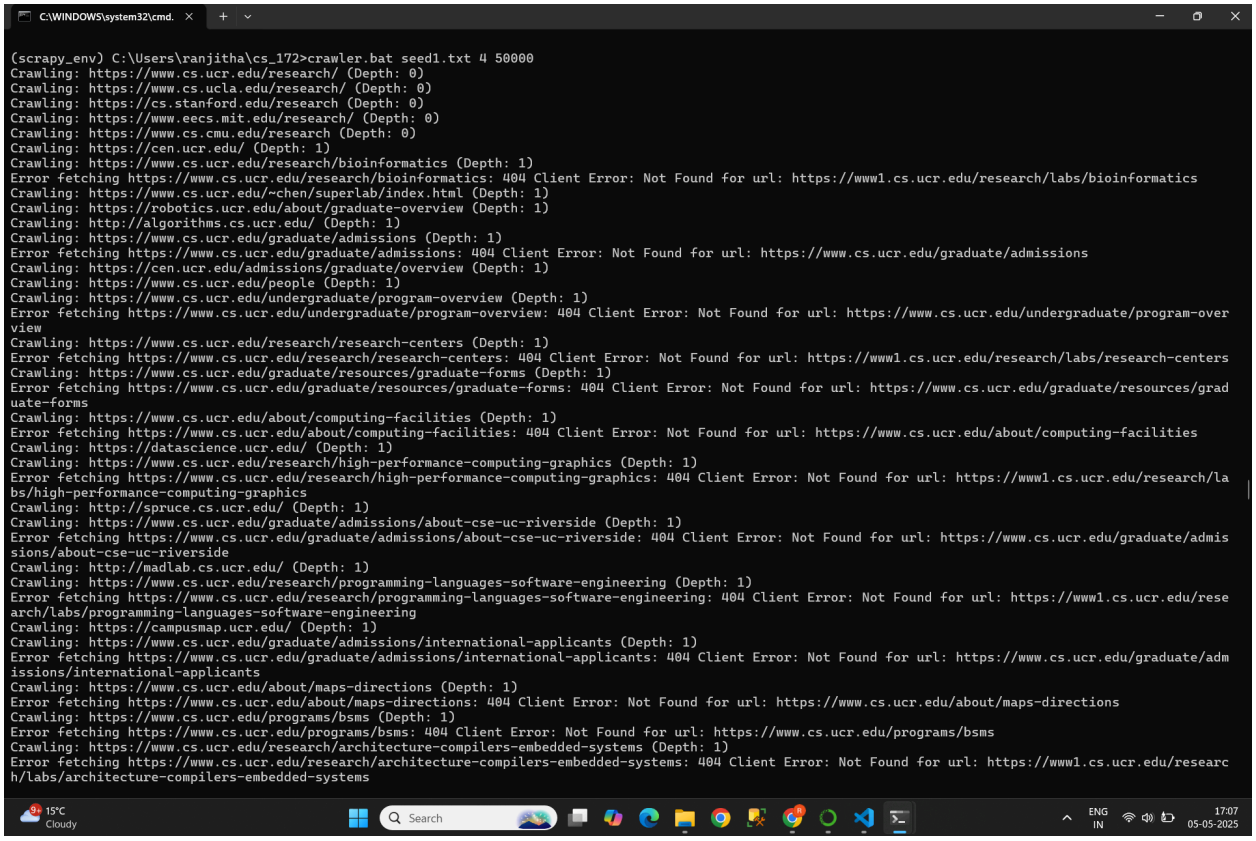

Figure 1: Crawler Execution in Windows CMD

Screenshot 2: Saved HTML Files

This screenshot displays the local folder (./data/html_pages/) where each crawled page is stored as a separate HTML file. Each filename is the SHA-256 hash of the normalized URL to ensure uniqueness and deduplication. The folder in the screenshot shows that over 5,000 HTML files were saved, indicating large-scale collection.

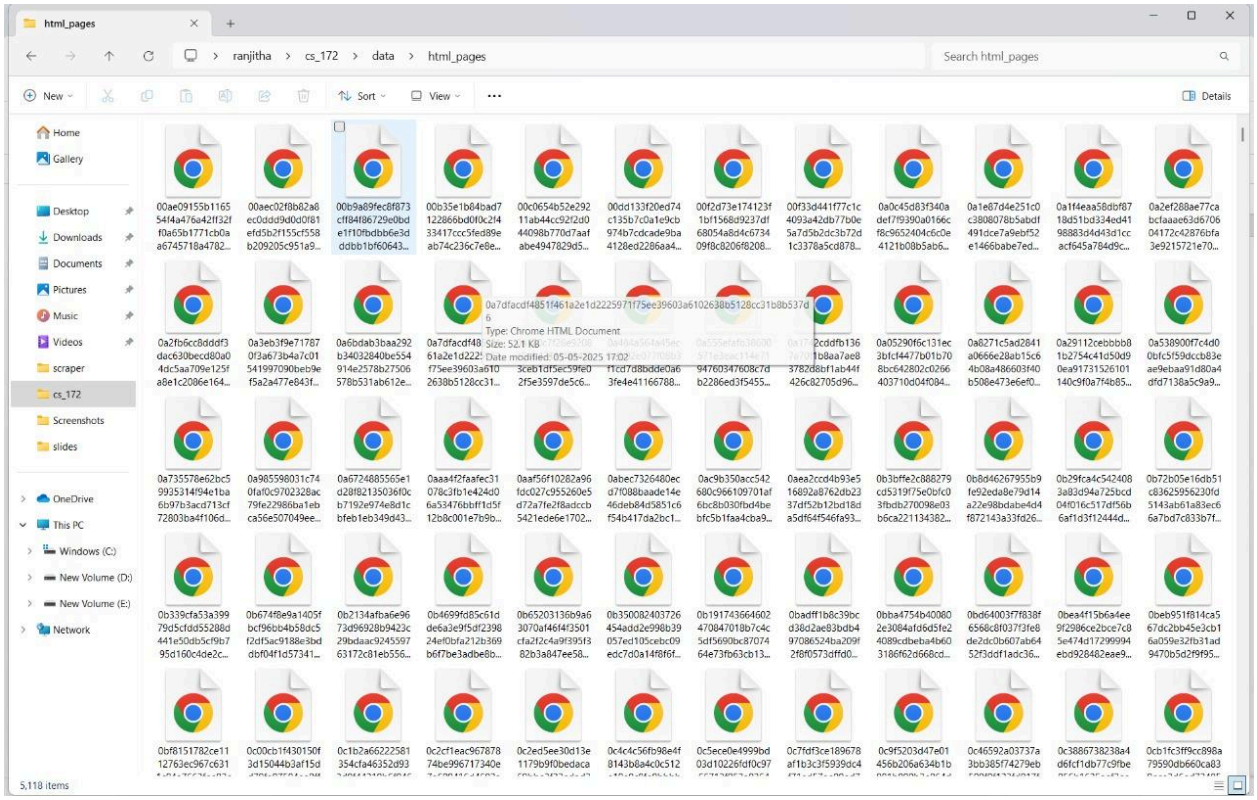

Figure 2: Output Directory with Saved Research HTML Pages

Limitations

- Single-threaded: Only one URL is fetched at a time (no parallelization).
- No robots.txt compliance: The crawler does not currently respect robots.txt files.
- No JavaScript rendering: Dynamic content that requires JS execution will be missed.
- No metadata extraction: Titles, authors, or structured tags are not parsed.
- No duplicate content detection across URLs (only URL-level deduplication is done).

Deployment Instructions

For Windows (crawler.bat):

1. Prepare a seed file (seed.txt) with one URL per line:
   https://www.cs.ucr.edu/research/
   https://www.cs.ucla.edu/research/
   https://cs.stanford.edu/research
   https://www.eecs.mit.edu/research/
   https://www.cs.cmu.edu/research
2. Run:
   crawler.bat seed.txt 4 50000

For macOS/Linux (manual or crawler.sh):

./crawler.sh seed.txt 4 50000

Directory Structure:

- crawler.py – main runner file to start crawling
- research_crawler.py – contains the ResearchPaperCrawler class
- seed.txt – input file with seed URLs
- data/html_pages/ – stores individual HTML pages as .html files
- crawler.bat – batch script for Windows
- visited – tracked internally via hashed URLs

Future Enhancements

- Add robots.txt parser to obey crawling rules.
- Implement multi-threading for faster crawling.
- Store visited hashes in a file for session persistence.
- Include HTML metadata (e.g., titles, headings) in output.
- Integrate with Lucene indexing in Part B.

Conclusion

This system demonstrates a domain-specific academic web crawler targeting .edu research pages. With a focus on clean architecture, breadth-first traversal, and structured HTML saving, the crawler achieves efficient and focused data collection.