CS-242

Information Retrieval & Web Search

Web Crawling and Indexing of Marvel and DC Fandom Pages

By:

- Aarav Babu: 862545619
- Aryan Ramachandra: 862545919
- Sankalp Naveenachandra Kulkarni: 862549757
- Sourav Guruprasad: 862546337
- Will Huang: 862234767

Link to Github repo

Contribution

Name	Architecture	Crawler Code	Data Collection	Indexing and Search	Report
Aarav Babu	✔️	✔️	✔️	✔️	
Aryan Ramachandra		✔️	✔️	✔️	
Sankalp Naveenachandra Kulkarni	✔️	✔️	✔️	✔️	
Sourav Guruprasad	✔️		✔️		✔️
Will Huang			✔️	✔️	✔️

Abstract

- This project focuses on developing a web crawler to collect data from Marvel and DC Fandom pages and indexing this data using PyLucene.
- The crawler extracts character-related content, processes the data for indexing, and allows for efficient searching and retrieval.
- The project aims to demonstrate the importance of structured web data extraction and full-text search capabilities using Lucene.
- By implementing a breadth-first search (BFS) based web crawler, we systematically retrieve information, filter redundant content, and process text for indexing.
- This allows for optimized search functionalities with accurate retrieval of superhero-related information.

Introduction

- Web crawling is a crucial technique for collecting data from the internet.
- It involves automated bots that navigate through web pages, extract relevant information, and store it for further processing.
- Indexing, on the other hand, is the process of organizing this data to enable efficient searching. This project aims to crawl and index character-related information from the Marvel and DC Fandom pages to build a structured repository for superhero data.
- Challenges such as duplicate content, site restrictions, and large-scale data handling are addressed through optimized crawling and indexing techniques.

System Architecture

1. Crawling System Architecture

Core Components

1. URL Management System

- Implementation of a queue-based crawler using collections.deque
- URL normalization using urllib.parse
- Duplicate URL detection via SHA-256 hashing
- Persistent storage of visited URLs in JSON format

2. Crawler Controller

- Depth-limited BFS implementation (configurable, default depth=2)
- Time-bound execution (configurable, default 3600 seconds)
- Session-based URL tracking
- Periodic data persistence

3. Content Extraction

- BeautifulSoup4 for HTML parsing
- Targeted content extraction focusing on paragraph elements
- URL filtering specific to the fandom.com domain

Crawling Strategy

1. URL Processing

- Normalized URL handling by removing fragments and query parameters
- Base URL preservation for relative link resolution
- Domain-specific filtering for fandom.com websites

2. Data Management

- Two-tier visited URL tracking:
    - ❖ Global visited URLs (persistent across sessions)
    - ❖ Session-specific visited URLs
- Periodic data saving (every 10 URLs)
- JSON-based storage format for both content and metadata

Code Snippet: A system architecture diagram showcasing the flow from crawling to storage.

```python
class WebCrawler:
    def __init__(self, seed_urls, max_depth=2, time_limit=3600, visited_file="../visited_urls.json", output_file="marvel1.json"):
        """   Initializes the WebCrawler with multiple seed URLs.
        - max_depth: BFS depth limit (default = 2)
        - time_limit: Max crawling time in seconds (default = 10 minutes)
        """
        # Load existing visited URLs but maintain a separate set for current session
        self.existing_visited = self.load_json(visited_file)
        self.current_session_visited = set()  # Track URLs visited in current session
        self.scraped_data = self.load_json(output_file)
        self.url_queue = deque([(url, 0) for url in seed_urls])  # Store (URL, depth)
        self.visited_file = visited_file
        self.output_file = output_file
        self.max_depth = max_depth
        self.time_limit = time_limit
        self.start_time = time.time()
```

```python
def crawl(self, url, depth):
    """
    Crawls a given URL: extracts text content, finds new links, and adds them to the queue.
    - Stops if max_depth is reached.
    """
    normalized_url = self.normalize_url(url)
    url_hash = self.generate_url_hash(normalized_url)

    # Check both existing and current session visited URLs
    if url_hash in self.existing_visited or url_hash in self.current_session_visited:
        return

    print(f"Scraping: {normalized_url} (Depth: {depth})")

    try:
        response = requests.get(normalized_url, timeout=10)
        response.raise_for_status()
    except requests.RequestException as e:
        print(f"Failed to fetch {normalized_url}: {e}")
        return

    soup = BeautifulSoup(response.content, 'html.parser')
```

Indexing System

The indexing pipeline consists of the following stages:

1. Text Preprocessing: Tokenization
2. Index Creation: Using PyLucene to build searchable indices.
3. Query Processing: Handling search queries efficiently.

Stop words were not removed because superhero names and affiliations often contain common words like "The", "Doctor", and "Captain." Removing them could reduce the accuracy of search queries. Keeping stop words improves recall and ensures precise phrase matching for entity-based searches.

Code Snippet: Illustrating how indexed data is stored and retrieved.

```python
import lucene
from org.apache.lucene.analysis.standard import StandardAnalyzer
from org.apache.lucene.document import Document, Field, TextField, StringField
from org.apache.lucene.index import IndexWriter, IndexWriterConfig
from org.apache.lucene.store import RAMDirectory

lucene.initVM()
directory = RAMDirectory()
analyzer = StandardAnalyzer()
config = IndexWriterConfig(analyzer)
writer = IndexWriter(directory, config)

doc = Document()
doc.add(StringField("title", "Spider-Man", Field.Store.YES))
doc.add(TextField("content", "Peter Parker is Spider-Man", Field.Store.YES))
writer.addDocument(doc)
writer.close()
```

Indexing Strategy

Fields in PyLucene Index

The PyLucene index is structured to store and retrieve superhero-related content efficiently. The key fields in the index include:

- Title: The character's name, indexed for easy retrieval.
- URL: The original source webpage for reference.
- Content: The extracted descriptions, character history, and abilities.
- Metadata: Additional attributes such as affiliations, origin, and notable features.

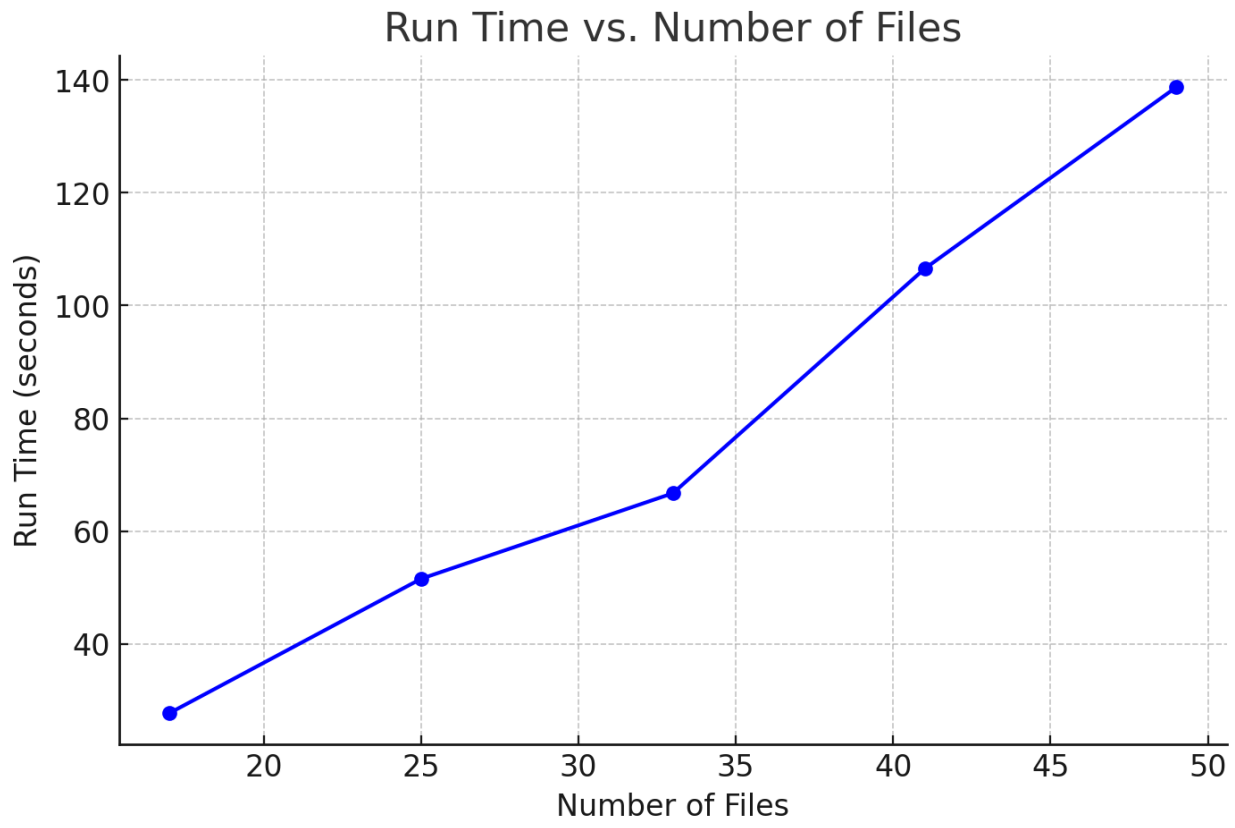
Each of these fields was selected to ensure comprehensive search capabilities while maintaining efficient storage.

Indexing Performance Analysis

A performance evaluation was conducted to measure the efficiency of the indexing process. The time taken to index documents was recorded against the number of documents processed.

Graph: A visual representation showing the runtime of index creation on the Y-axis against the number of documents on the X-axis.

Screenshot: Included below to illustrate performance results.



Search Functionality

The search functionality allows users to retrieve superhero-related information efficiently using Lucene's query parser.

Example Queries:

1. Attribute Query: *Shang Chi*

```
Indexing Statistics:
2025-02-14 19:30:27,013 - INFO - Total documents: 197942
2025-02-14 19:30:27,013 - INFO - Successfully processed: 197942
2025-02-14 19:30:27,013 - INFO - Failed documents: 0
2025-02-14 19:30:27,013 - INFO - Total time: 137.65 seconds
2025-02-14 19:30:27,013 - INFO - Average processing rate: 1437.97 docs/second
2025-02-14 19:30:27,026 - INFO -
Executing search for: 'Shang Chi'
2025-02-14 19:30:27,026 - INFO - Index contains 197942 documents
```

Performing Searches

```
indexer.search("Shang Chi")
indexer.search("Spider-Man") #, fields=["basic_info.name", "basic_info.aliases"])
```

```
Indexing Statistics:
2025-02-14 20:11:11,036 - INFO - Total documents: 197942
2025-02-14 20:11:11,037 - INFO - Successfully processed: 197942
2025-02-14 20:11:11,037 - INFO - Failed documents: 0
2025-02-14 20:11:11,037 - INFO - Total time: 139.34 seconds
2025-02-14 20:11:11,037 - INFO - Average processing rate: 1420.55 docs/second
2025-02-14 20:11:11,054 - INFO -
Executing search for: 'Venom'
2025-02-14 20:11:11,054 - INFO - Index contains 197942 documents
```

Searching the indexed structure:

```
cs242@class-043:~/fandom/fandom/spiders/test/aarav$ python3 lucene_query.py
Loading index from directory: marvel_index
Enter search query: Shang Chi

Executing search for: 'Shang Chi'

Search Results:
==================================================

Match Score: 10.18
--------------------------------------------------
ID: acb8565a096972e17690e623c9fa6496f048f77a10b7c265872dda7d30d74bba
URL: https://marvelcinematicuniverse.fandom.com/wiki/Shang-Chi
Content Snippet: Are you currently rewatchingDaredevilleading up toDaredevil: Born Again? Consider participating in ourRoad to
Born Again Editing Eventas we strive to polish up our older articles!
READ MORE
Shang-ChiReal NameXu Shang-Chi (徐尚气)Alias(es)Shaun[1]In-Universe MediaBus Boy[1]SpeciesHumanCitizenshipChinese

==================================================

Match Score: 10.09
--------------------------------------------------
ID: 81d37942a50ed5967d28be04fed04a0ce5d52246223fb47aad720646df00f770
URL: https://marvel.fandom.com/wiki/Zheng_Shang-Chi_(Earth-13116)
Content Snippet: Shang-ChiGalleryNameZheng Shang-ChiAliasesEditorial Names:Master of Kung FuAffiliation and RelationshipsAffili
ationLowest CasteFormerlyTen RingsRelativesZheng Zu(father)Marital StatusSinglePhysical CharacteristicsGenderMaleEyesIrises:Bro
wnHairBlackOrigin and Living StatusOriginHumanLiving StatusAliv

==================================================
```

Deployment Instructions

[Instructions to Run Code]

Indexer Configuration

```
indexer = LuceneIndexer(
    max_workers=4,
    commit_batch=1000,
    ram_buffer_size_mb=256
)
```

Running Multiple File Processing

```
json_files = [f"marvel{i}.json" for i in range(1, 17)]
json_files.append("marvel.json")
```

```
2025-02-14 19:30:23,372 - INFO - Processed 191915/197942 documents
2025-02-14 19:30:23,969 - INFO - Processed 192915/197942 documents
2025-02-14 19:30:24,370 - INFO - Processed 193915/197942 documents
2025-02-14 19:30:24,948 - INFO - Processed 194915/197942 documents
2025-02-14 19:30:25,242 - INFO - Processed 195915/197942 documents
2025-02-14 19:30:25,533 - INFO - Processed 196915/197942 documents
2025-02-14 19:30:25,836 - INFO - Processed 197915/197942 documents
2025-02-14 19:30:25,870 - INFO - Processed 197942/197942 documents
2025-02-14 19:30:25,870 - INFO - Completed processing ../scraper/data/marvel_will8.json
```

```
# Process files
indexer.index_documents(valid_files)
```

Challenges & Solutions

Handling Duplicate Pages

Implemented URL hashing to store visited links and prevent re-crawling.

Optimizing Crawling Speed

- Used multi-threading (if applicable) to parallelize requests.
- Introduced request delay to avoid blocking by the site.

Managing Large-Scale Data Storage

- Stored extracted data in JSON format.
- Indexed only relevant text content to optimize disk usage.

Avoiding Request Blocking from Fandom

- Implemented user-agent rotation.
- Limited request rates to comply with site policies.

Conclusion & Future Work

This project successfully developed a web crawler that efficiently scrapes superhero data from Fandom pages and indexes it using PyLucene. The implementation demonstrates structured web data extraction and indexing, enabling efficient search capabilities. Future work can explore:

- Handling JavaScript-rendered content using Selenium.
- Expanding the dataset to include real-time updates.
- Enhancing search ranking algorithms with AI-powered relevance scoring.
- Images
- Text Summary of Top Results