CS 242: Information Retrieval & Web Search

Winter 2020

Team members: Brandon Evans, Silvia Cabellos, William Mac, Jasper Sandhu, Yue Liu.

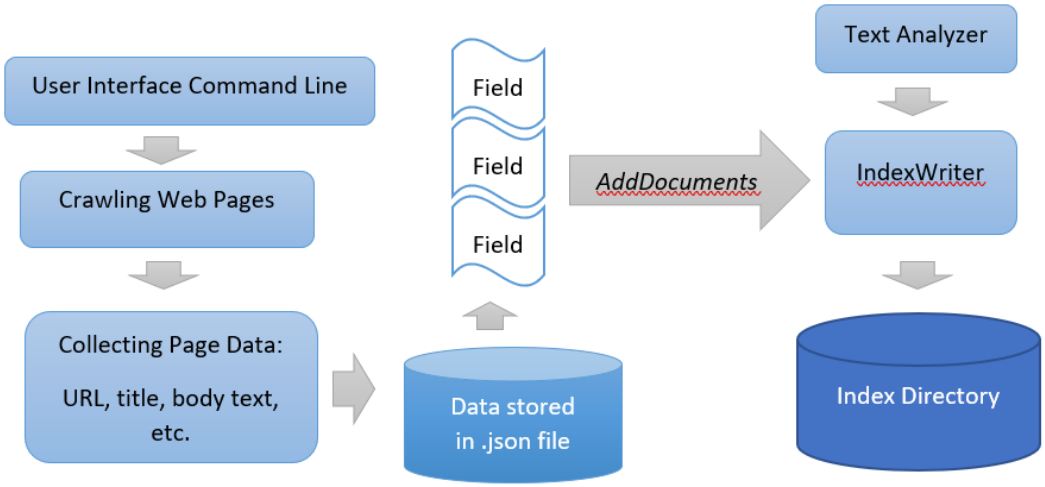
Project Report Part A: Data Collection and Indexing with Lucene

Collaboration Details: Description of the contribution of each team member.

Crawling: Silvia Cabellos, Brandon Evans
Indexing: Yue Liu, Jasper Sandhu, Brandon Evans
Reporting/Coding: William Mac
Testing : All members

Project Overview :

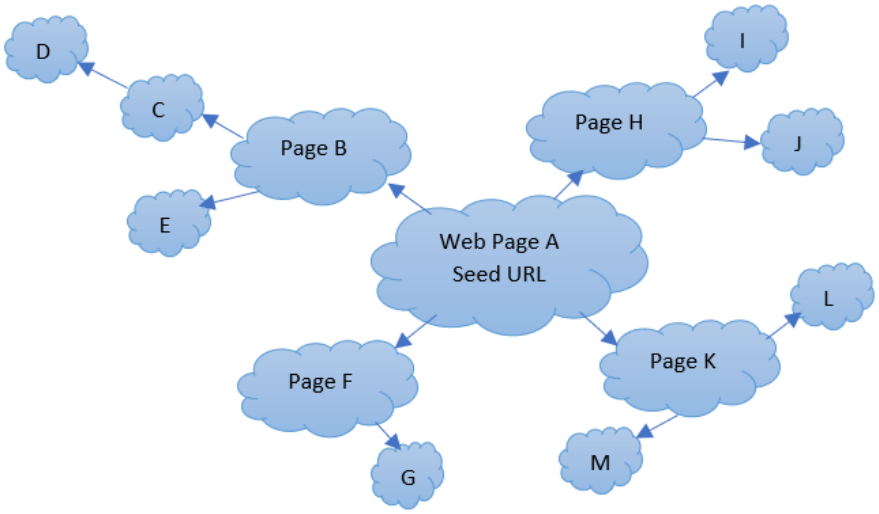
The scope of this project was a webcrawler that could "crawl" across one or more seed websites and extract their links, subsequently crawling sites to a given recursion level or depth. After the sites are crawled and desired attributes extracted, we implement a Lucene-based indexer to efficiently analyze and index the crawled attributes into a format that can easily be searched. The application of crawling and indexing could be useful for internal company websites, or for fast information access to given topics on the public Internet.
Widespread, webcrawling and indexing are used by the top search engines, like Google, and others.



Crawling system:

Architecture and Strategy:

For this part of the project we designed and built a jsoup based web crawler. The strategy for the crawler would be to use a recursive function to start crawling from the seed URL and go into the links within a certain depth set by the user. By default, depth level is set to 3, but we can change it depending on the website we are trying to get data from.
The crawler gets the following elements from the websites: page URL, page title and body text and links. We are also getting metadata (description and keywords), which eventually will help us with the search engine



Crawling Sequence:
A - B - C - D - E …etc.

Within the crawler, we included the following classes:

Webcrawler : Define the crawling strategy
CrawlerData: Determines the format and naming convention of file to store all the data. The name convention chosen is a simple DateTime format. The crawler is designed to save the data crawled into .json files (16 MB). The idea is to get a few files storing all the data.

CrawlerPageData : State all the data type that's being collected by the crawler.
Main: Set the command-line input options and default argument values for crawling and indexing. Print help page.

Efficiency of the crawler:
Avoid duplicate pages: Duplicates are preventing using a hashmap in Java to store previously-crawled pages

Maximizing performance: We use an in-memory class to store page results up to a size threshold and periodically write JSON files (target 16MB).

Threading: We quickly recognized that multi-thread would improve performance, but did not have sufficient time to configure for our implementation. A ThreadPool that processed a queue would have been an ideal solution.

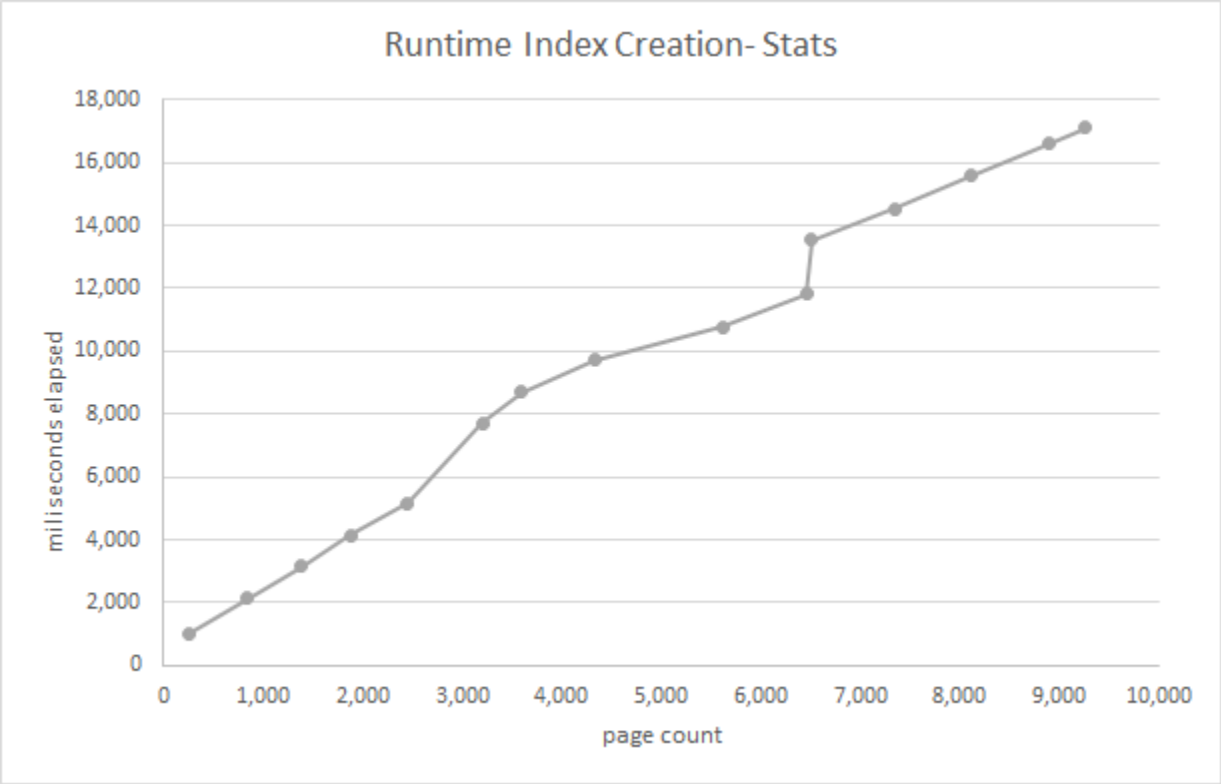
Using the crawler, we collected 1.17GB of data.

Overview of the Lucene indexing strategy:

Indexed Fields : The fields included in the Lucene index are “Url”, “title” and “body” because they are the most basic and searchable elements in HTML structure

Text Analyzer: We used StandardAnalyzer which is the most commonly used analyzer. The reasons for choosing StandardAnalyzer are because it can recognize URLs and lowercase the generated token. Additionally, StandardAnalyzer is able to remove the Stop Words. We decided to add a corpus with 405 common words to remove common terms from web documents and implement a custom Stop Word List. This list can be edited by the user.

Index Creation Chart: Below is a chart showing run time on the Y-axis compared to the number of documents indexed. Notice there is a point where the index was slower than usual, taking around 2 seconds to process around 40 pages.



Limitations (if any) of the system.

During this initial project, we found the following limitations

- Crawler skipping over pages heavy with Javascript. One of the limitations we discovered while using JSoup to scrape html pages, JSoup or any variation of a simple html parser isn't able to scrape javascript generated html. Additional tools are required to run to allow for this process.

Obstacles and solutions.

- Issues iterating through crawler data pages (e.g. crawler not traversing back from final depth). We discovered our depth-first approach would be better served by a breadth-first crawling strategy, architected with a queue to hold recursive (child) URLs, but keeping the crawler work to incremental levels.

- Originally used Lucene 8.4 (Latest) but reverted to an older version of Lucene. Mainly because newer versions of Lucene didn't seem to have enough tutorials and code examples to follow. Attempting to use the latest release (8.4) the challenges of integrating all the methods to interact with each other were very cumbersome that resulted in countless hours resolving a single process. The solution was to use a slightly older version (Lucene Core 7.1).

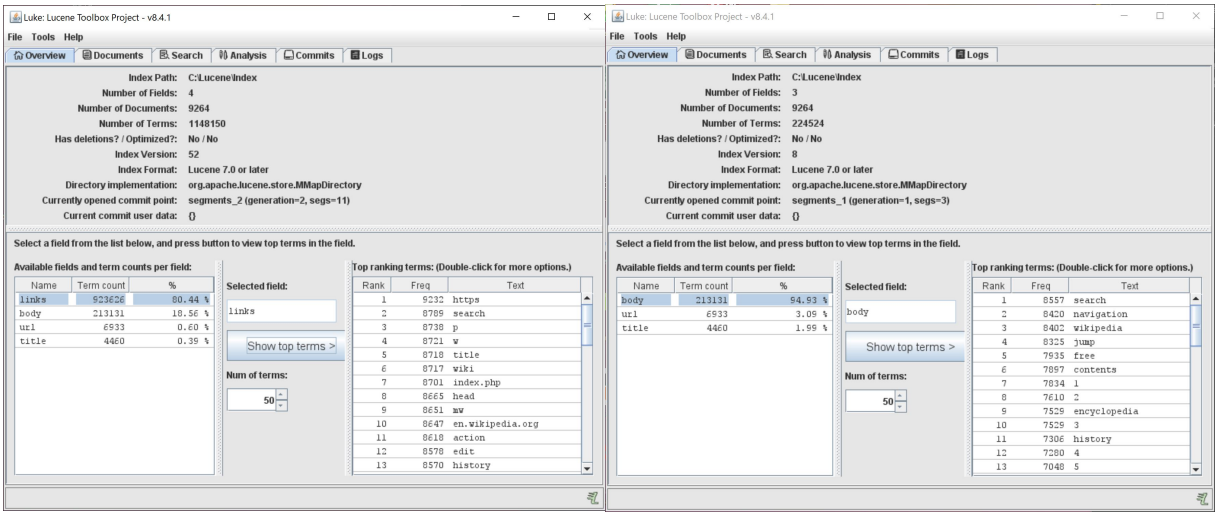
- Lack of an effective Stop Word Filter built-in Lucene. The default immutable list consisted of 33 immutable words that were not sufficient for the type of data we're crawling.
  The solution was to use the most commonly used stop words that consist of 405 words from "https://www.lextek.com/manuals/onix/stopwords1.html" as well as implement a way that the user could easily edit on the fly depending on their needs. We Added a method within Indexer Class that reads a "StopWordList.txt" that resides in the root of the program folder.

- Issues when trying to crawl websites without much metadata. Not all sites have metadata, we approached trying to obtain meta description and keywords but found we weren't getting consistent results, so questioned the importance of the data in the index.

- Issues when trying to crawl popular websites resulted in rejection from the website with no errors. We were able to overcome this by editing the "User-Agent" parameter to tell the web server we're using Chrome Browser Application.

- Seeing HTTP errors with various status codes when crawling some web pages. We added error handling code to ignore all pages that returned HTTP exceptions.

- Including Links into the Indexer as <TextFields> seems to have a negative effect when the user queries the Index. Generally it's likely the user is more interested in the terms within the body of the article.



Left Image = Indexing Links, **Right Image** = Excluding Links from the Index. Noting the Term counts.

Instruction on how to deploy the crawler.

The deliverables included executable BAT file and JAR file for references. However, the full path of java command was specified in BAT file, in case your java command path is different, it's preferred to deploy the crawler and indexer using below syntax:
java -jar IR_CS242.jar {ARGUMENTS}

You can refer to the following detailed usage instruction:

Usage Instructions
The Java application is a console application, meant for use at the command-line/terminal of your Linux/Mac/Windows PC.

You'll want to have Java version 11 on your PC.

General Usage:
java -jar IR_CS242.jar [-c] [-cd <arg>] [-h] [-ir] [-iw] [-oc <arg>] [-oi <arg>] [-s <arg>] [-t <arg>]

Options

-c,--crawl	Crawl Mode
-cd,--crawlDepth <arg>	Max crawl depth (default 5)
-h,--help	Show Help
-ir,--indexRead	Index Read Mode
-iw,--indexWrite	Index Write Mode
-oc,--crawlData <arg>	Crawler Output Folder
-oi,--indexData <arg>	Index Folder
-s,--seed <arg>	Seed Url
-t,--term <arg>	Search Term

Crawler Example:

java -jar IR_CS242.jar -c -cd 3 -oc c:\MyFolder -s https://en.wikipedia.org/wiki/Kobe_Bryant

(-c enables crawler mode) (-cd means crawl depth of 3 levels) (-oc specifies where to output the crawler data - JSON) (-s specifics the seed/root URL to start crawling)

Instruction on how to build the Lucene index.

Index Writing Example:
java -jar IR_CS242.jar -iw --oi C:\MyIndexFolder -oc c:\MyDataFolder

(-iw enables index writing mode) (-oc specifies where to read the crawler data - JSON) (-oi specifies where to output the Lucene index data)

Index Reading Example (EXPERIMENTAL)
java -jar IR_CS242.jar -ir --oi C:\MyIndexFolder -t “search this”

(-ir enables index read/search mode) (-oi specifies where to output the Lucene index data) (-t is the search term)

References

The following references and articles were used during this project.

http://ignaciosuay.com/getting-started-with-lucene-and-json-indexing/
https://www.baeldung.com/lucene
https://jsoup.org/cookbook/