TEAM MEMBERS

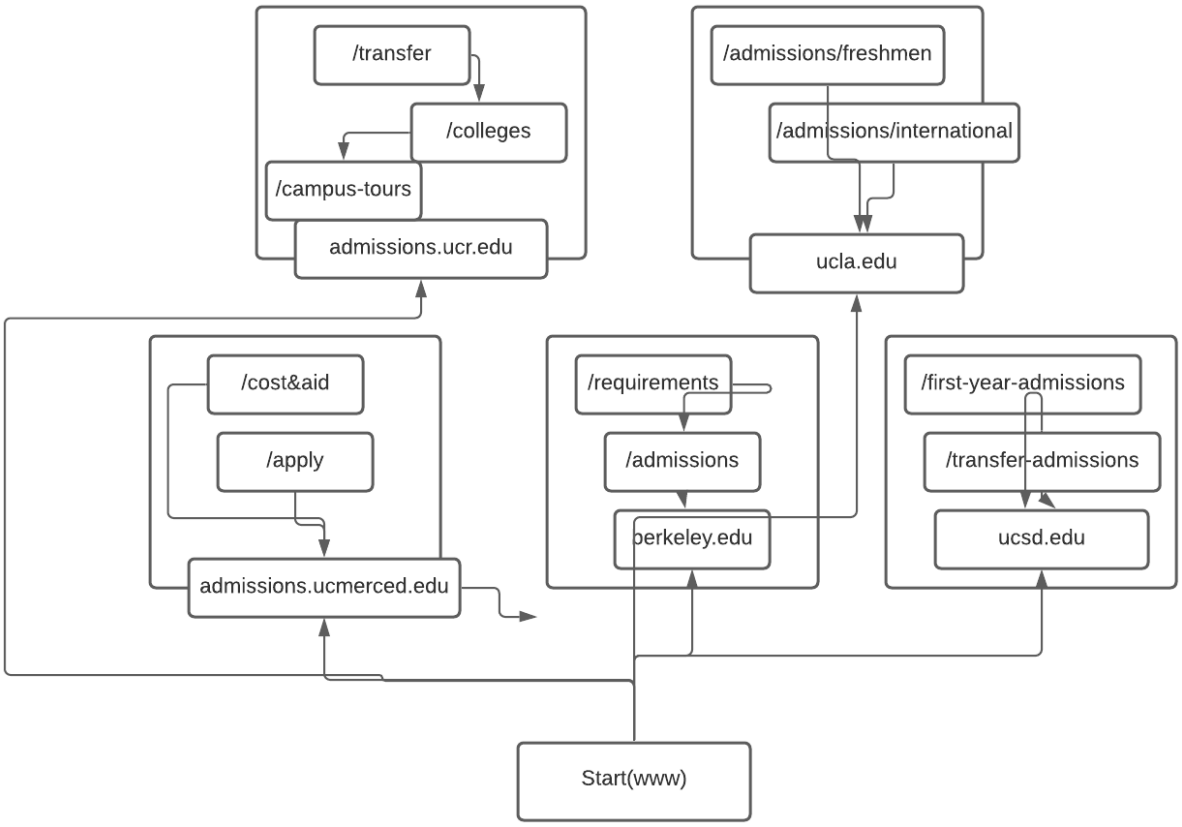
Name	SID	NetID	Collaboration
Mohamed Rayyan	862037325	mrayy002	• Crawler • Crawling System Overview • Lucene deployer
Douglas Adjei-Frempah	862259280	dadje001	• Report Crawler & Lucene Architecture • Graph • Lucene deployer • Web interface(PART B)
Abhijeet Ramesh Murthy	862325105	amurt004	• Lucune code functionality • Testing code • Parser functionality to parse the data file
Yashwanth Eshwarappa	862313223	yeshw001	• Lucune code functionality • Testing code • Parser functionality to parse the data file

PROJECT: BUILDING A SEARCH ENGINE FOR ADMISSION INFORMATION OF VARIOUS COLLEGES

Crawling System Overview

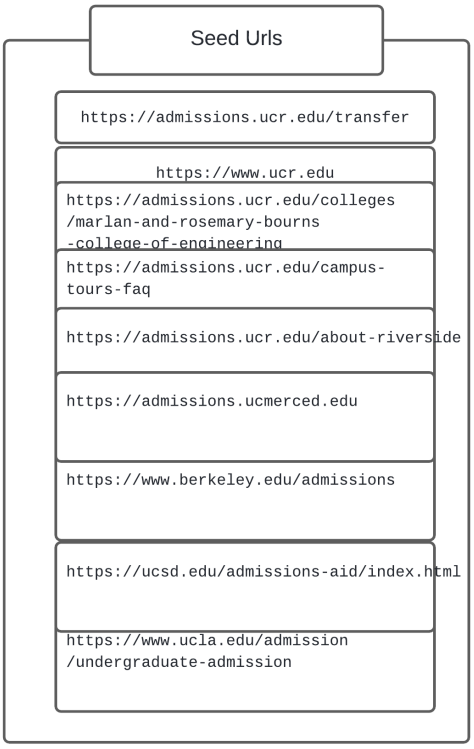
Architecture

The crawler contains multiple methods and attributes that significantly contribute to the success of the crawler.



- **Methods**
  - extractAndMergeLinks(List<Element> els, List<String> links, List<String> skipURL, HashMap<String, Boolean> visited)
  - removeStropWords(String body, List<String> stopWords)
  - isValidURL(String url)
  - savePages(List<String> pages)

- **Attributes**
  - Int dataSize
  - List<String> links
  - List<String> stopWords
  - List<String> pages
  - List<String> skipURL
  - HashMap<String, Boolean> visited
  - HashMap<String, Boolean> visitedHash



Crawling Strategy

The crawler begins its journey by crawling urls from a seed file that contains about 18 urls. Before retrieving a web page the crawler checks whether the url has been visited before by checking a hashmap where the key is the url and the value is boolean. If the url has been visited the crawler simply moves on to the next url in the list. However, if the url has not been visited before, the crawler retrieves the webpage and filters text to only human readable text in the body tag <body>. After the body text has been extracted, stop words are removed from the text and the text size in bytes is added to the attribute **dataSize** (contains the crawled data size in bytes). Additionally, the crawler calls the method **extractAndMergeLinks** where all the urls in the page are extracted. Before the extracted urls are appended to the **links** attribute each url is checked whether it has been visited before, already in **links** or if the url is in the **skipURL** attribute. After the urls are appended to **links**, the crawler marks the crawled url as visited additionally the body content of the page is fed into a sha256 function which returns a hash and stored in the **visitedHash** map which is another method to eliminate duplicates. Finally, the crawler repeats the mentioned above instructions for the next url in the list.

Lucene Overview

Standard Analyzer

We opted for the standard analyzer because the most important thing we needed to recognize was URLs, we also wanted some tokenization to break the contents of our crawls into sentences. We also wanted the added convenience of removing stop words that appear

multiple times in all the other documents so that their mere presence and occurrence does not skew the rankings in their favor. Since stop words occur more than more important and meaning-carrying words, many ranking models would assign an unreasonable level of importance to them if we allow them to appear in our data to be passed to the various ranking models. The standard analyzer also allows us to convert all entries into lowercase so that case is not really a differentiator between otherwise similar or same words.

Indexed Fields in Lucene

Title: this is the title of the crawled webpage.

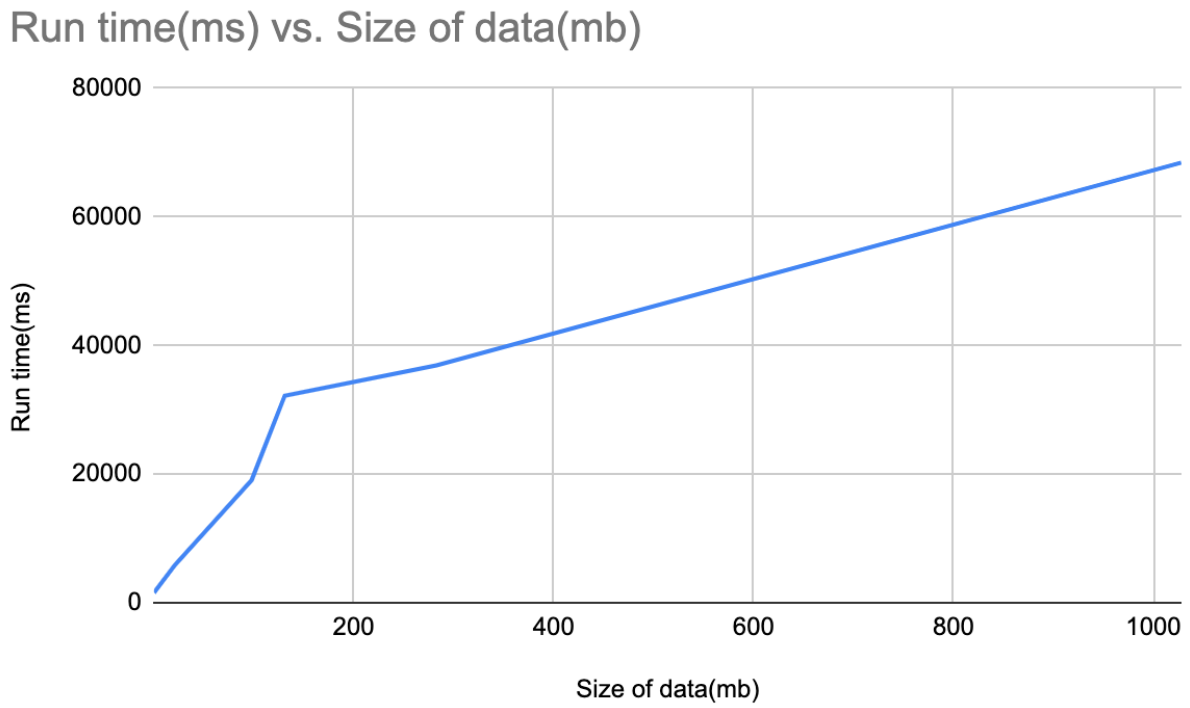
Sub Headers (<h3> tag): these are subheaders found in webpage that indicate an important topic

Body text: this is all readable text from a webpage body. This excludes html tags, links and images.

Lucene Run Time

We crawled and had different amounts of data during the crawling process. So we tested the various file sizes with creating the lucene index and the run times for those files were as follows:

Size of data(mb)	**Run time(ms)**
0.319	1570
21	5820
98	19080
131	32190
283	36900
1028	68450



Limitations

The main limitations of the crawler is that it's not multithreaded. This limitation causes the crawler to be slow when the required crawled data size is in the gigabytes range.

Obstacles and Solutions

One major obstacle faced during the execution of the crawler is that many urls either return 400, 404, 500, or 503 error codes. This becomes an issue when the same urls appear again in the list of urls to crawl which leads to the crawler being slow. To resolve this issue when a url returns an error code, that url is appended to skipURL, thus preventing that url from being attempted to crawl.

Another obstacle faced during the execution of the crawler is that many different urls that point to the same page thus causing duplicates. Although we have a hashmap named visited where the key is the url that has been crawled, that alone did not stop duplicates. Thus we have implemented an additional hashmap named **hashVisited** where the key is the output hash of the **sha256** algorithm and the input is the crawled body text from the page.

Deploying Lucene Instructions

Assuming you're in the root directory of the submission directory
Usage as follows;

```
cd Lucene/
./deploy.sh 319kb.dat
```