Hopkins Search Engine

Team members

Jiaxuan Zhang (601.666, jzhan239@jhu.edu), Yiyang Li (601.666, yli302@jhu.edu)

Project summary:

Web and CLI apps to help Hopkins students gain easier access to the latest events, activities, and announcements on campus and within the Hopkins community. We allow users the options to (1) quickly query pre-crawled data we update periodically, or (2) crawl the latest info as they search. Both the web and CLI interface exposes many options (e.g., which high-level domain to search from, which similarity function to use, etc.) for the users to tweak that allows for fine-grained search control.

GitHub repository: https://github.com/robertzhidealx/cs466-project

Set up

Create and enter into a Python virtual environment, and then install dependencies. For more information and for the Windows commands, please refer to the official docs.

```
python3 -m venv .venv
. .venv/bin/activate
pip install flask flask-cors numpy nltk bs4
```

`deactivate` to exit the virtual environment.

Crawler

Run the crawler to prepare the pre-crawled data. The crawler is engineered to extract information from websites within domain www.jhu.edu, hub.jhu.edu and www.jhunewsletter.com more effectively, while also flexible enough to crawl from other websites.

To run the crawler by itself, use `python webCrawl.py {url}` (url should be without the https:// prefix) to crawl websites with default settings. This would generate {url}.latest.txt containing 1000 crawled pages, with 4 lines per page. The first line contains the url, the second line contains the title, the third line contains the author, and the fourth line contains the body text. Our pre-crawled data resides in versions that don't have the ".latest" suffix, e.g., `hub.jhu.edu.txt`.

Crawling could also be done from the web app, generating {url}.latest.txt files, again with the format specified previously.

Search Engine

Run the CLI app via `python3 query.py` or from within the venv, `python query.py`.

To run the Web app, first enter the venv and start the backend server:

```
. .venv/bin/activate
flask run
```

The server is now running and will be used to serve data to the frontend. The server file is app.py.

To start the frontend website, open a new terminal and navigate to the `jhu-search-engine` directory.

This is a JavaScript (Next.js) project, so in order to run it, you need to first install *the latest* version of Node.js (>= v18.17.0) and NPM (the Node.js package manager) by following the instructions here.

Once done, first install dependencies by running:

```
npm install
```

Then, start the frontend by running:

```
npm run dev
```

Navigate to `http://localhost:3000` in your browser. Voila!

jhu-search-engine/src/pages/index.js is the main web page file for the web frontend.

Let us know at any point if you have issues running our project. Thank you very much!

Achievements

- Crawling is optimized to ensure no duplicate access, and in-domain search is ensured so we get info only from our desired websites.
- Information extraction is engineered to accurately access desired information from JHU websites, increasing efficiency and accuracy for JHU related searches.
- Queue and stack strategies for crawling are implemented, so user could use different crawling strategies corresponding to different websites. For both implementations, we skip certain subtrees of pages due to their not offering much useful articles to users. And for a similar reason, we do not weight pages based on i.e. the number of slashes in their URL since this can lead to either mostly higher-level pages (non-articles) getting retrieved first (thus slow to get to articles) or pages that are too specific (also non-articles).
- Our web interface is modern and well-designed, providing users with a range of options to facilitate deriving the most relevant pages. Pages retrieved are ranked from the most relevant to the least relevant, whether it's the web app or the CLI app. The search button on the web app gets disabled when crawling is in progress, thus preventing the user from starting another crawl when the current one has not finished. This follows best practices in design. The retrieved entries additionally show their index in all the pages crawled and their corresponding URL for easy reference of the actual page.

- We give the users maximum flexibility by not only providing the JHU Hub and the JHU Newsletter as options to crawl and search from, but also arbitrary domains as specified by users. Again, they may do this on our intuitively designed web interface via a dropdown menu.
- We provide multiple modailities of interaction with our tool, both through a web graphical user interface and through a CLI tool. For users more accustomed to GUIs, the web app is the way to go; for those who want a quick search, the CLI tool can be a good option.
- On the web app, users may choose if they want to search using the pre-crawled dataset or crawl the latest data on the fly. The pros and cons of each option are obvious: with the former, info retrieval is much faster but data may not be the most up to date; with the latter, you get the most recent events but at the cost of much slower runtime - crawling 100 pages take 30s to a minute, on our end.
- We designed our tool based on the client-server architecture, with clear-cut APIs bridging the two. As a result, we are able to use a common set of functions to serve the web frontend and the CLI frontend, including functions related to info retrieval and those related to web crawling.
- We allow search pages based on their title, author, and body text (which is part of the main body text), and additionally select whether to remove stopwords, stem words, compute term weights using IDF, TF-IDF, or Boolean, compute similarity using cosine, jaccard, dice, or overlap, which relative weighting scheme to use across the three fields, and how many top pages to retrieve.
- On the web app, if a user wants to crawl on the fly, they may specify the number of pages they want to the crawler to stop with, depending on how much time they have on their hands.
- In our CLI app, the user is guided through a series of questions to determine their preferred configuration of the search. Most of these steps have a default selected option (e.g., cosine, TF-IDF) and the user only needs to press enter to select the default, instead of having to manually enter them each and every time they try to search something. User experience is what we care about!

Limitations

- Crawling and information extraction from other websites might not be as effective, due to variations in website designs. That said, we tried our best to come up with a general purpose HTML search scheme to find fields like the author of an article. To achieve higher precision, one may try to increase the max word limit and the number of pages to crawl for non-JHU websites.
- Our web app UI can be improved by the addition of a paging system, instead of having the user scroll all the way down for certain entries.
- A perma-link from within our web app can be provided for each retrieved entry, for easier sharing with others. This feature will require very careful engineering thanks to the dynamic nature of our search engine. We didn't explore this due to time constraints.
- It takes a while to crawl pages, thus it would be a nice extension to parallelize this process to give users real-time feedback while crawling is in progress.

Screenshots & Empirical evaluation

Since our tool is essentially a search engine, there is no obvious metric by which we can evaluate its effectiveness against real-time data. Thus, we take a qualitative approach to empirically evaluate our work.

In the screenshots directory, we have included a collection of screenshots corresponding to different scenarios and use cases of our tool. Below is a list describing their effectiveness:

- admit_student_kirkman shows the top results of a search based on the latest live-crawled data that seeks for "admit" in the title, "student" in the body text, and "kirkman" as the author. As you can see, the most relevant result in this case meets all three criteria. Since we're using the 1,1,4 weighting scheme where 4 is assigned to the body text, it is more important to have "student" in the body text than to have "admit" in the title. This is reflected in the remaining results shown, which *all* have "student" in the body text.

- admit_student_kirkman_latest is a version of the previous search run on pre-crawled data. This leads to slightly more historical articles compared with the previous use case.

- commence_romney searches for pages that has "commence" in the title and "romney" in the body text. As you can see, the top result has both commencement in the title and Mitt Romney mentioned in the body text. The second and third page have Romney in the body text but not commencement in the title. The fourth and fifth page have commencement in the title but no mention of Romney in the body text. This ranking makes total sense and would most likely fit the user's information need.

- romney_commence flips the importance of the two criteria in the previous search, where "commence" being in the body text is more important. As a result, article 4 (with Romney in the title) went from rank 1 to rank 3, which makes sense.

- commence_rosen_114 searches pages that have "Commence" in the body text and "rosen" as the author. 4/5 of the shown results mention commencement in their body text. In particular, Jill Rosen is the author of the third page. The reason why this page is not ranked higher is that we are using the 1,1,4 weighting scheme where body text outweighs author by a huge margin. We will observe a similar search next. The last page has Rosen as the author but not on commencement (truncated in this image but shown in a related article in the next one).

- commence_rosen_111 searches pages that have "Commence" in the body text and "rosen" as the author, but with a uniform weighting scheme of 1,1,1. As a result, it makes sense that the article Rosen wrote on commencement is ranked first instead of third now. Page 31, ranked last, also has Rosen as the author but not on commencement. Again, we see rankings that are exactly consistent with our expectations.

- loading shows the UI when the latest pages are being crawled live as part of a search. Notice how the button is disabled while "loading..." is shown instead of the usual "Crawl & Search".

- romney_2024_newsletter shows a search from the JHU Newsletter for "romney" in the title and "2024" in the body text. The results again make sense, where the first two results both have Romney in the body text, and the third result has "2024" in the title but no Romney in the body text.

- cli shows an interative session of our CLI app, where the user searches for pages whose title mentions "2024" and body text mentions "romney". The result is exactly as we would expect.

Notes

Besides the Python/JavaScript libraries we utilize, all code was written by the two of us (based to some extent on skeletons provided in prior assignments of this course).