COMP 4601A
Fall 2023 – Assignment #1

Objectives

The goal of this assignment is to implement a basic search engine using the concepts and tools covered in the first half of the course. To complete this assignment, you will need to implement a web crawler, a RESTful server, and a browser-based client that will allow a user to perform searches. You will also need to deploy your server to OpenStack and integrate it with a distributed search service.

Submission Requirements

There will be several requirements for submission:

1. The code for your assignment must be submitted on Brightspace. You should not submit your database files. This submission must include a README file with your name, any partner's names (if applicable), a summary of the parts of the assignment that you did/didn't complete successfully, and a link to your video demonstration. **If submitting as a group, only one member should make a submission.**
2. Your assignment must also be deployed to OpenStack so the TA can execute search requests. Include the URLs the TA should use to query your search engine in your README file.
3. You must record a <15 minute video where you:
    a. Demonstrate your search engine's functionality for both crawled sites.
    b. Discuss the design of your implementation (see the end of the document for some ideas of things you could discuss).
    c. Discuss the personal site you selected, the challenges that it presented, and how you addressed these challenges.
4. Your video demonstration must be hosted online. You can use any tools/host you want. Carleton offers support for recording and hosting videos using Kaltura and MediaSpace. See https://carleton.ca/kaltura/personal/record-with-personal-capture/ and https://carleton.ca/kaltura/add-new-media/ for information. If you are using MediaSpace, you can set your video to 'Unlisted' and share the link. Ensure the video can be viewed even if you are not logged into your account.

Assignment Requirements

The web crawler portion of your assignment must be capable of crawling the following:

1. The fruit example site. Start at people.scs.carleton.ca/~davidmckenney/fruitgraph/N-0.html and crawl the entire site (1000 pages).
2. Another site of your choosing. Limit the total number of crawled pages to 500-1000. It is suggested to limit your crawl within the same domain that you begin. You can design your selection policy to focus on any pages or resources you deem important. You are not required to crawl non-HTML resources but can choose to do so.

Your crawled data must be stored in a database for persistence. Your crawler must also perform PageRank calculations and store the values for each page in the database.

Your RESTful web server must read the data from the database, perform required indexing, and provide relevant, ranked search results for any valid request. Your server must support GET requests for at least the following endpoints:

1. /fruits – represents a request to search the data from the fruit example
2. /personal – represents a request to search the data in the alternate site you selected

Both of your search endpoints (/fruits and /personal) must support all combinations of the following query parameters:

1. q – A string representing the search query the user has entered, which may contain multiple words.
2. boost – Either true or false, indicating whether each page should be boosted in the search results using its PageRank score.
3. limit – A number specifying how many results the user wants returned (minimum 1, maximum 50, default 10). If a valid limit parameter *X* is specified, your server MUST return *X* results, even if all documents have a score of 0 (return any *X* documents in this case).

The browser-based interface for searching must allow the client to specify:

1. The text for their search
2. Whether they want the results to be boosted or not using PageRank
3. The number of results they want to receive (minimum 1, maximum 50, default 10)

The search results displayed in the browser must contain:

1. The URL to the original page
2. The title of the original page
3. The computed search score for the page
4. The PageRank of the page within your crawled network
5. A link to view the data your search engine has for this page. This must include at least the URL, title, list of incoming links to this page, list of outgoing links from this page, and word frequency information for the page (e.g., banana occurred 6 times, apple occurred 9 times, etc.). You can also display any additional data you produced during the crawl.

If a search request specifies that JSON data should be sent in the response, the response should be a JSON string containing an array of the most relevant *X* results, ranked from highest to lowest score. Each object entry in this array should contain the following keys:

1. name – A string containing the names of all group members
2. url – The URL of the original page
3. score – The page's computed search score for this search request
4. title - The title of the original page
5. pr - The PageRank of the page within your crawled network

In order to connect to the distributed search engine, after your server has started, it should send a PUT request to http://134.117.130.17:3000/searchengines that contains JSON data (make sure you include the Content-Type header to indicate it is JSON data). The request body should have the format {request_url: 'URL_for_your_server'}, where 'URL_for_your_server' represents the base URL of your deployed server (i.e., a GET request to URL_for_your_server/fruits would search fruits and a GET request to URL_for_your_server/personal would search your personal site).

You can use the axios library or a similar option to make the request. If your request is formatted well, the URL of your server is correct, and you have followed the assignment specification, you should receive a response with status 201 (created) or a message indicating that the specified URL has already been registered. At this point, your search server has been added into the list of available search engines. If something went wrong, you should get a 40x error code and a message indicating some information about why the request was unsuccessful.

If you want to double-check the status of your own server (or any other registered server), you can make a GET request to http://134.117.130.17:3000/searchengines which will list all of the registered servers. This page will also indicate if the last search result was successfully handled by each server. If you want to test a distributed search, you can make a GET request to either http://134.117.130.17:3000/fruits?q=YourQueryHere or http://134.117.130.17:3000/personal?q=YourQueryHere , which will retrieve search results from all operational registered search engines, merge the results, and send a response. Each server is given 2 seconds to send a response.

Potential Discussion Points

Below are some things to consider addressing in your demonstration video. This list is not exhaustive, so include any other meaningful analysis you think is valuable.

1. How does your crawler work? What information does it extract from the page? How does it store the data? Is there any intermediary processing you perform to facilitate the later steps of the assignment?
2. Discuss the RESTful design of your server. How has your implementation incorporated the various REST principles?
3. Explain how the content score for the search is generated.

4. Discuss the PageRank calculation and how you have implemented it.
5. How have you defined your page selection policy for your crawler for your personal site?
6. Why did you select the personal site you chose? Did you run into any problems when working with this site? How did you address these problems?
7. Critique your search engine. How well does it work? How well will it scale? How do you think it could be improved?