HW #1 - Web Science Intro

Ethan Landers

Due: September 15, 2024

Q1

Consider the "bow-tie" structure of the web in the Broder et al. paper *"Graph Structure in the Web"* that was described in Module 1.

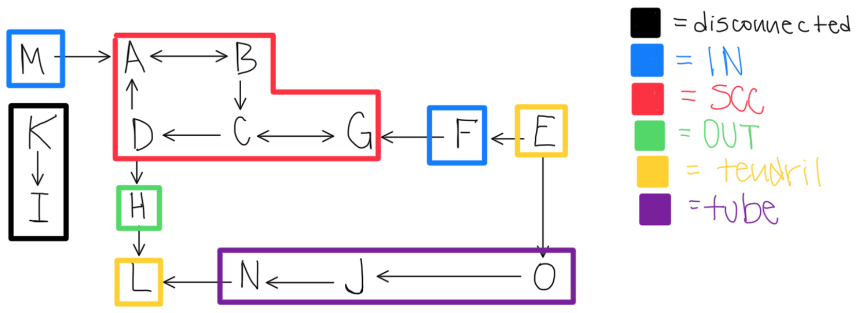
Now consider the following links:

```
A --> B
B --> A
B --> C
C --> D
C --> G
D --> A
D --> H
E --> F
E --> O
F --> G
G --> C
H --> L
J --> N
K --> I
M --> A
N --> L
O --> J
```

Draw the resulting directed graph (either sketch on paper or use another tool) showing how the nodes are connected to each other and include an image in your report. This does not need to fit into the bow-tie type diagram but should look more similar to the graph on slide 24 from Module-01 Web-Science-Architecture.

For the graph, list the nodes (in alphabetical order) that are each of the following categories:

Answer



- **SCC:**
  - A
  - B
  - C
  - D
  - G
- **IN:**
  - F
  - M
- **OUT:**
  - H
- **Tendrils:**
  - E
    - ∗ Node E can reach the IN Node F.
  - L
    - ∗ Node L can reach the OUT Node L.
- **Tubes:**

  These nodes connect tendril E to tendril L:
  - J
  - O
  - N

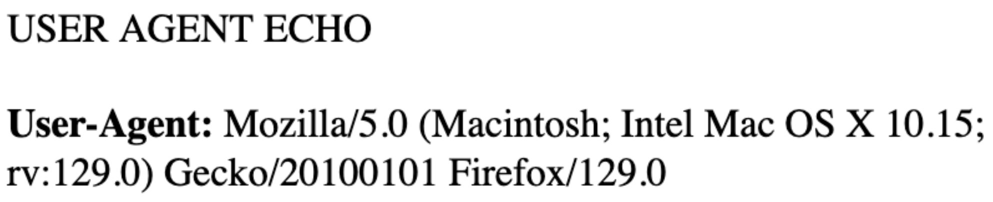
- **Disconnected:**
  - I
  - K

Q2

Demonstrate that you know how to use curl and are familiar with the available options. Complete the following steps using https://www.cs.odu.edu/ mweigle/courses/cs532/ua_echo.php as the URI to request. Explain the results you get from each step.

Answer

1. First, load the webpage at the URI in your web browser. The result should show the "User-Agent" HTTP request header that your web browser sends to the web server. Take a screenshot to include in your report.



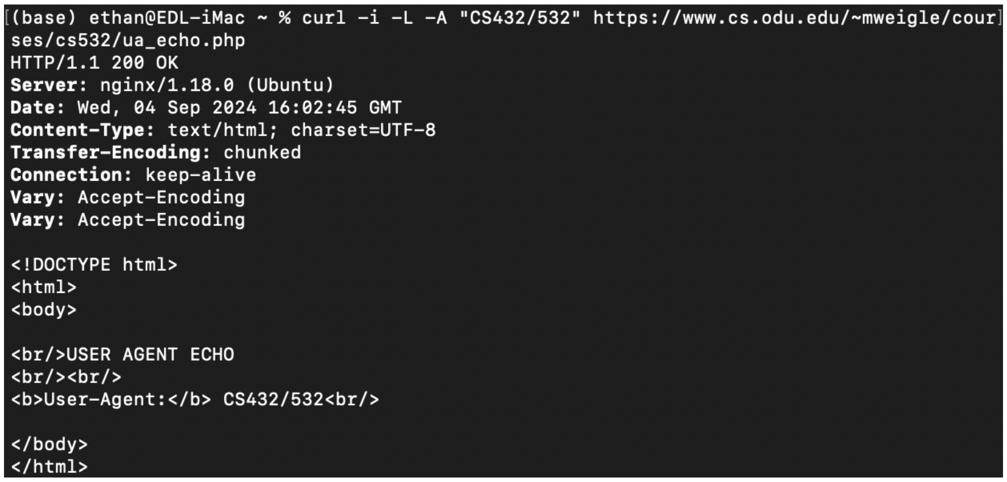
USER AGENT ECHO

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:129.0) Gecko/20100101 Firefox/129.0

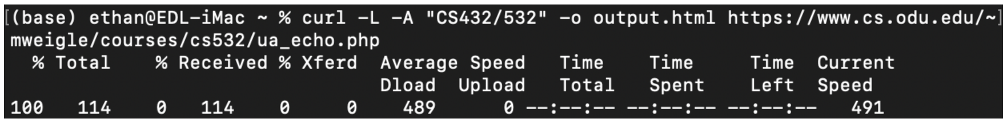
2. Use a single curl command with the appropriate options to do the following:
   - request the URI
   - show the HTTP response headers
   - follow any redirects
   - change the User-Agent HTTP request field to "CS432/532"

   Take a screenshot of the curl command and options you used and the result of your execution to include in your report.

3. Use a single curl command with the appropriate options to do the following:
   - request the URI
   - follow any redirects
   - change the User-Agent HTTP request field to "CS432/532"
   - save the HTML output to a file



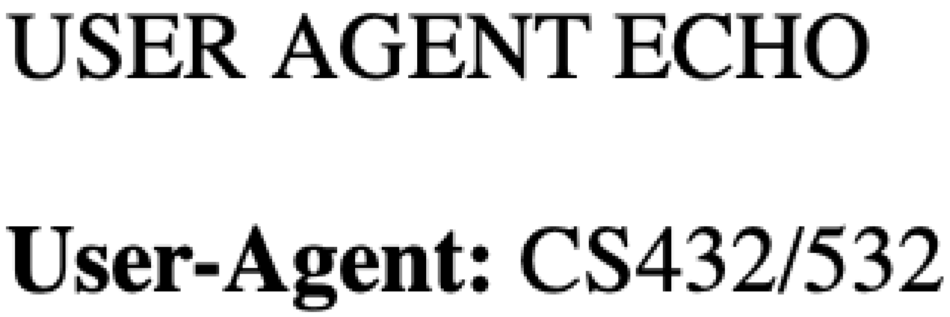
```
(base) ethan@EDL-iMac ~ % curl -i -L -A "CS432/532" https://www.cs.odu.edu/~mweigle/cour
ses/cs532/ua_echo.php
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Wed, 04 Sep 2024 16:02:45 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Vary: Accept-Encoding
Vary: Accept-Encoding

<!DOCTYPE html>
<html>
<body>

<br/>USER AGENT ECHO
<br/><br/>
<b>User-Agent:</b> CS432/532<br/>

</body>
</html>
```



```
(base) ethan@EDL-iMac ~ % curl -L -A "CS432/532" -o output.html https://www.cs.odu.edu/~
mweigle/courses/cs532/ua_echo.php
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   114    0   114    0     0    489      0 --:--:-- --:--:-- --:--:--   491
```

Take a screenshot of the curl command and options you used and the result of your execution to include in your report.

4. View the HTML output file produced by curl from part c in a web browser and take a screenshot to include in your report.



USER AGENT ECHO

User-Agent: CS432/532

Q3

Your program must do the following:

1. take the URI of a seed webpage as a command-line argument
2. extract all the links from the page's HTML
3. for each link, request the URI and use the Content-Type HTTP response header to determine if the link references an HTML file (text/html)
    - if it does, use the Content-Length HTTP response header to determine if it contains more than 1000 bytes
        - if it does, then print the final URI (after any redirects)

Use this program to collect at least 500 unique URIs of web pages that contain more than 1000 bytes. Save the list of URIs to a file to use in later assignments. The file must be uploaded to your GitHub repo.

Answer

1. I created a check to ensure that the user provided a command-line seed URI argument when running the program:

```
    if __name__ == "__main__":
    # Ensure the user provided a command-line argument
    if len(sys.argv) != 2:
        print("Usage: python3 collect-webpages.py <seed_uri>")
        sys.exit(1)

    # Start the main process with the provided seed URI
    main(sys.argv[1])

```

   If the user doesn't provide a command-line argument, a message will appear directing the user to the correct way to run the program, which would include a command-line argument.

2. I created a function to extract all links from a given webpage:

```
def extract_links_from_page(uri):
    print(f"Fetching page: {uri}")
    try:
        response = requests.get(uri, timeout=5, verify=False)

        # Parse the HTMl content using BeautifulSoup
```

```python
        soup = BeautifulSoup(response.text, 'html.parser')
        links = set()

        for link in soup.find_all('a', attrs={'href': re.compile("^https://")}):
            href = link.get('href')
            if href:
                links.add(href)

        print(f"Found {len(links)} links on page")
        return links

    except requests.exceptions.RequestException as e:
        print(f"Error fetching {uri}: {e}")
        return set()  # Return an empty set if there is an error

```

The requests library makes HTTP GET requests to the provided URI. BeautifulSoup parses HTML pages and extracts links that start with HTTP://. The links are then returned as a set to ensure uniqueness. The code includes a few debug statements.

3. I created a function to verify if links are valid HTML pages:

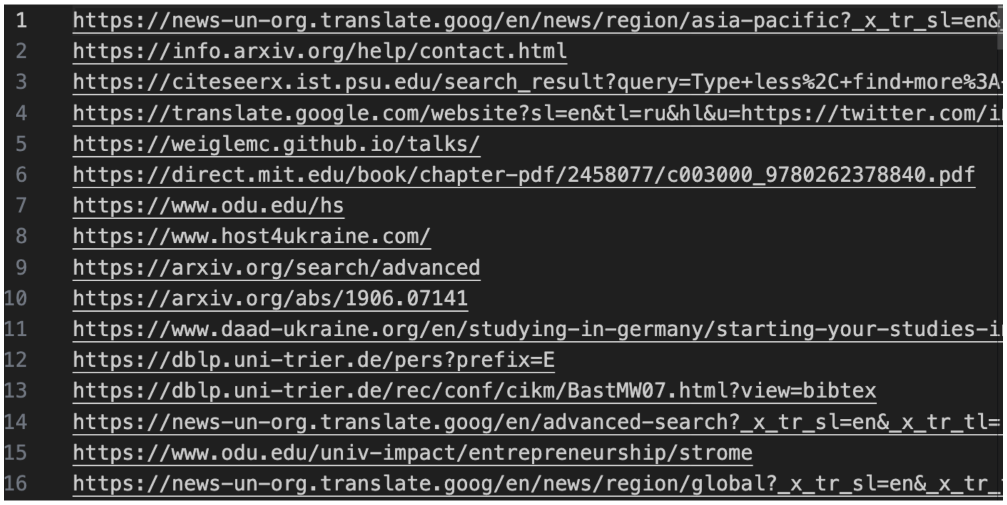
```python
def is_valid_html_page(uri):
    print(f"Validating URI: {uri}")
    try:
        response = requests.head(uri, timeout=5, verify=False)
        content_type = response.headers.get('Content-Type', '')
        content_length = response.headers.get('Content-Length', 0)

        if 'text/html' in content_type and int(content_length) > 1000:
            print(f"Valid HTML page: {uri}")
            return True
        else:
            print(f"Invalid HTML page or too small: {uri}")
            return False

    except requests.exceptions.Timeout as e:
        print(f"Timeout error validating {uri}: {e}")
        return False
    except requests.exceptions.RequestException as e:
        print(f"Error validating {uri}: {e}")
        return False

```

This function returns boolean variables by checking HTML headers to determine whether they are valid HTML pages and whether they have more than 1000 bytes. Exception states are printed otherwise.

After the HTML pages are verified, they will be added to a set of URIs and printed to an output file called uris.txt.

The program randomly picks a URI from the provided seed URI and uses that as the new seed until 500 unique URIs are collected.

Here is a screenshot of what a part of the output file uris.txt looks like:



```
https://news-un-org.translate.goog/en/news/region/asia-pacific?_x_tr_sl=en&
https://info.arxiv.org/help/contact.html
https://citeseerx.ist.psu.edu/search_result?query=Type+less%2C+find+more%3A
https://translate.google.com/website?sl=en&tl=ru&hl&u=https://twitter.com/i
https://weiglemc.github.io/talks/
https://direct.mit.edu/book/chapter-pdf/2458077/c003000_9780262378840.pdf
https://www.odu.edu/hs
https://www.host4ukraine.com/
https://arxiv.org/search/advanced
https://arxiv.org/abs/1906.07141
https://www.daad-ukraine.org/en/studying-in-germany/starting-your-studies-i
https://dblp.uni-trier.de/pers?prefix=E
https://dblp.uni-trier.de/rec/conf/cikm/BastMW07.html?view=bibtex
https://news-un-org.translate.goog/en/advanced-search?_x_tr_sl=en&_x_tr_tl=
https://www.odu.edu/univ-impact/entrepreneurship/strome
https://news-un-org.translate.goog/en/news/region/global?_x_tr_sl=en&_x_tr_
```

References

- Overleaf documentation, `https://www.overleaf.com/learn`
- Curl documentation, `https://curl.se/docs/manpage.html`
- Python Virtual Environments, `https://python.land/virtual-environments/virtualenv`
- Python Data Types, `https://www.geeksforgeeks.org/python-data-types/`
- Implementing Web Scraping with BeautifulSoup, `https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/`
- BeautifulSoup Scraping Link from HTML, `https://www.geeksforgeeks.org/beautifulsoup scraping-link-from-html/?ref=header_outind`
- Exception Handling in Python Requests Module, `https://www.geeksforgeeks.org/exception-handling-of-python-requests-module/`
- Python Random Choice Method, `https://www.w3schools.com/python/ref_random_choice.asp`

- Writing to File in Python, `https://www.geeksforgeeks.org/writing-to-file-in-python/`