# HW 1 - Web Science Intro

Adeniran Adeniyi

Sunday, Feb 7th 2021 by 11:59pm

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

*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: You may copy the question into your report, but make sure that you make it clear where the question ends and your answer begins.*

- SCC:
- IN:
- OUT:
- Tendrils:
  - indicate if the tendril is reachable from IN or can reach OUT
- Tubes:
  - explain how the nodes serve as tubes

## Answer

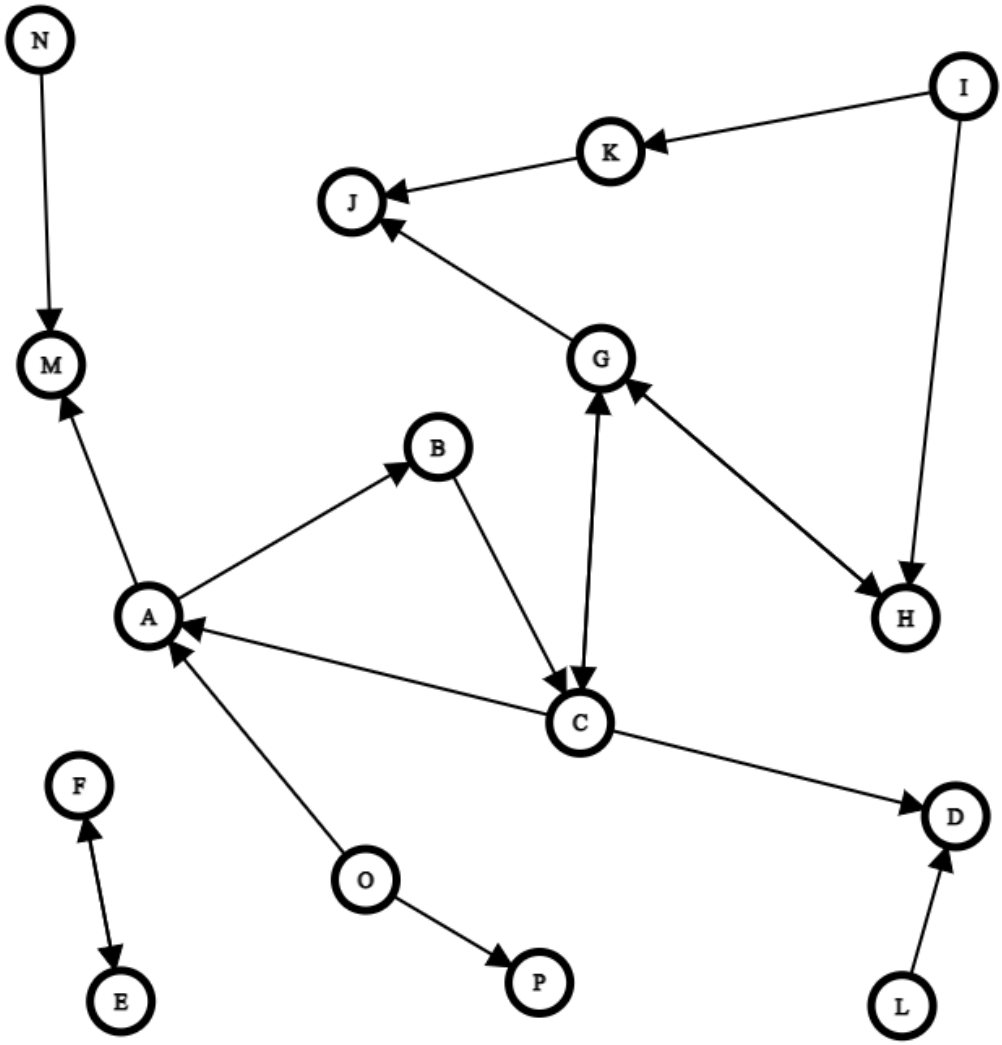


**Figure 1:** Directed graph Question 1

- SCC: A, B, C, G, H
- IN: O, I
- OUT: M, D, J
- Tendrils: N, L, P
- Tubes: K
- Disconnected: E, F

## Discussion

*I followed these instruction below:*

- IN: Pages with no in-links or in-links from IN pages and out-links to pages in IN, SCC, Tendrils, or OUT (via a Tube)
- SCC: Pages with in-links from IN or SCC and out-links to OUT or SCC. There exists some path of links from every page in SCC to every other page in SCC
- OUT: Pages with no out-links or out-links to other pages in OUT, and all in-links come from OUT, SCC, Tendrils, or Tubes.
- Tendrils: Pages that can only be reached from IN or have only out-links to OUT.
- Disconnected: Pages that have no in-links from any other components and no out-links to other components. These pages may be linked to each other.

# Q2

*Demostrate that you know how to use curl and are familiar with the available options.*
*URL to request:* `http://www.cs.odu.edu/~mweigle/courses/cs532/ua_echo.php`

(a) First, load the URI directly in your browser and take a screenshot. The resulting webpage should show the User-Agent HTTP header that your web browser sends to the web server.

(b) In a single curl command, request the URI, show the HTTP response headers, follow any redirects, and change the User-Agent HTTP request field to ”CS432/532”. Show command you used and the result of your execution on the command line. (Either take a screenshot of your terminal or copy/paste into a code segment.)

(c) Then make the same request again, but without showing the HTTP response headers and with saving the HTML output to a file (i.e., change the options you provide to the curl command). Show the command you used and the result of your execution on the command line. View the HTML output file that was produced by curl in a web browser and take a screenshot.

*Explain the results you get for each of these steps.*

## Answer

*(a)*

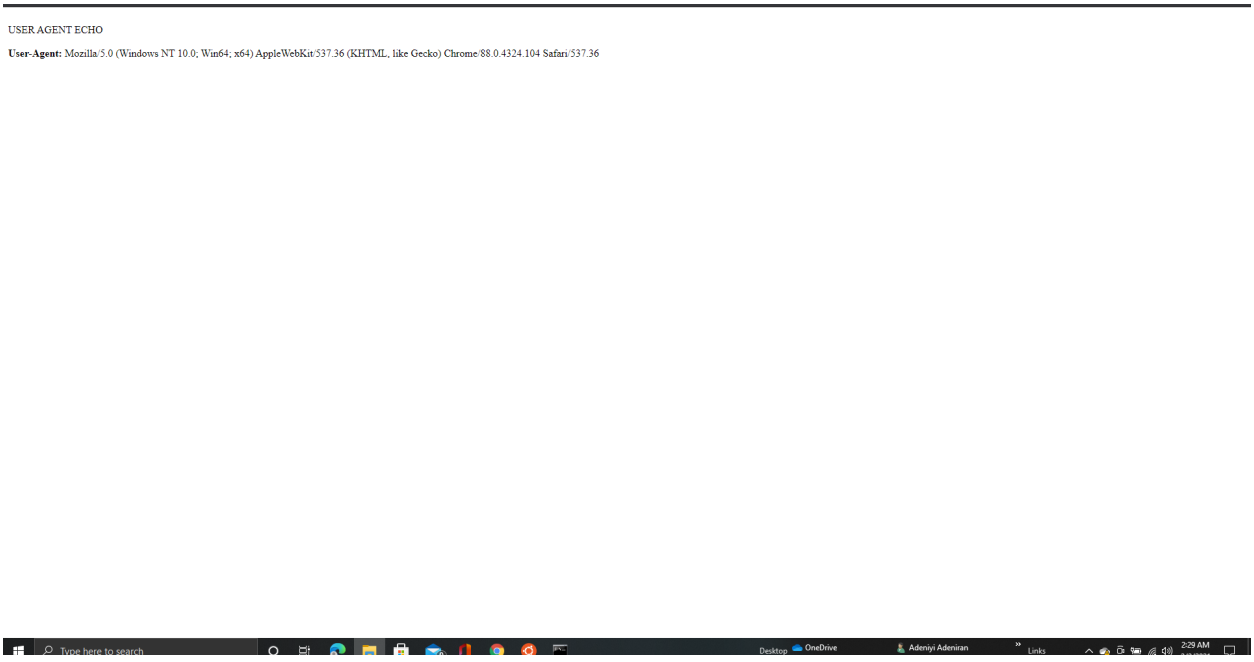


**Figure 2:** Question 2(a) URL in Browser output

## Discussion

*The User Agent displayed on the browser after clicking the* `http://www.cs.odu.edu/~mweigle/courses/cs532/ua_echo.php` *took me to a new webpage that displayed the output shown bellow.*
*User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.146 Safari/537.36*

*(b)*

```
curl --head -L http://www.cs.odu.edu/~weigle/courses/cs532/ua_echo.php --next -A "CS432/532" http://www.cs.odu.edu/~mweigle/cs532/ua_echo.php
```

```
HTTP/1.1 301 Moved Permanently
Server: nginx
Date: Sat, 06 Feb 2021 17:59:48 GMT
Content-Type: text/html
Content-Length: 178
Connection: keep-alive
Location: https://www.cs.odu.edu/~weigle/courses/cs532/ua_echo.php

HTTP/1.1 302 Found
Server: nginx
Date: Sat, 06 Feb 2021 17:59:49 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 0
Connection: keep-alive
Allow: GET, HEAD, OPTIONS
Location: https://www.cs.odu.edu
Referrer-Policy: same-origin
X-Content-Type-Options: nosniff
X-Frame-Options: DENY

HTTP/1.1 301 Moved Permanently
Server: nginx
Date: Sat, 06 Feb 2021 17:59:50 GMT
Content-Type: text/html
Content-Length: 178
Connection: keep-alive
Location: https://odu.edu/compsci

HTTP/1.1 200 OK
Date: Sat, 06 Feb 2021 17:59:51 GMT
Server: Apache/2.4.6 (Red Hat Enterprise Linux)
Vary: Host
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Connection: close
Content-Type: text/html; charset=UTF-8
Set-Cookie: BIGipServerWEB_HTTPS_PROD.app~WEB_HTTPS_PROD_pool_int=rd741o00000000000000000ffff8052619eo80; path=/; Httponly; Secure

<html>
<head><title>301 Moved Permanently</title></head>
<body bgcolor="white">
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx</center>
```

```
</body>
</html>
```

## Discussion

*I Entered the command on the terminal area*

```
curl --head -L http://www.cs.odu.edu/~weigle/courses/cs532/ua_echo.
php --next -A "CS432/532" http://www.cs.odu.edu/~mweigle/cs532/
ua_echo.php
```

*Where -A stands for user-agent <name> Send User-Agent <name> to server according to curls help command "curl –help". This allows me to specify a custom user agent which I used "CS432/532" as a substitute. I used the option "-L" which is usually called location is to make curl follow redirects. The –next is used to execute another command in one. The user-agent for both is different. This first one used curl/7.68.0 while the other connection used CS432/532. The –head option displays the header*

*(c)*

```
curl -L http://www.cs.odu.edu/~weigle/courses/cs532/ua_echo.php --next
-A "CS432/532" http://www.cs.odu.edu/~mweigle/cs532/ua_echo.php -o
index.html
```

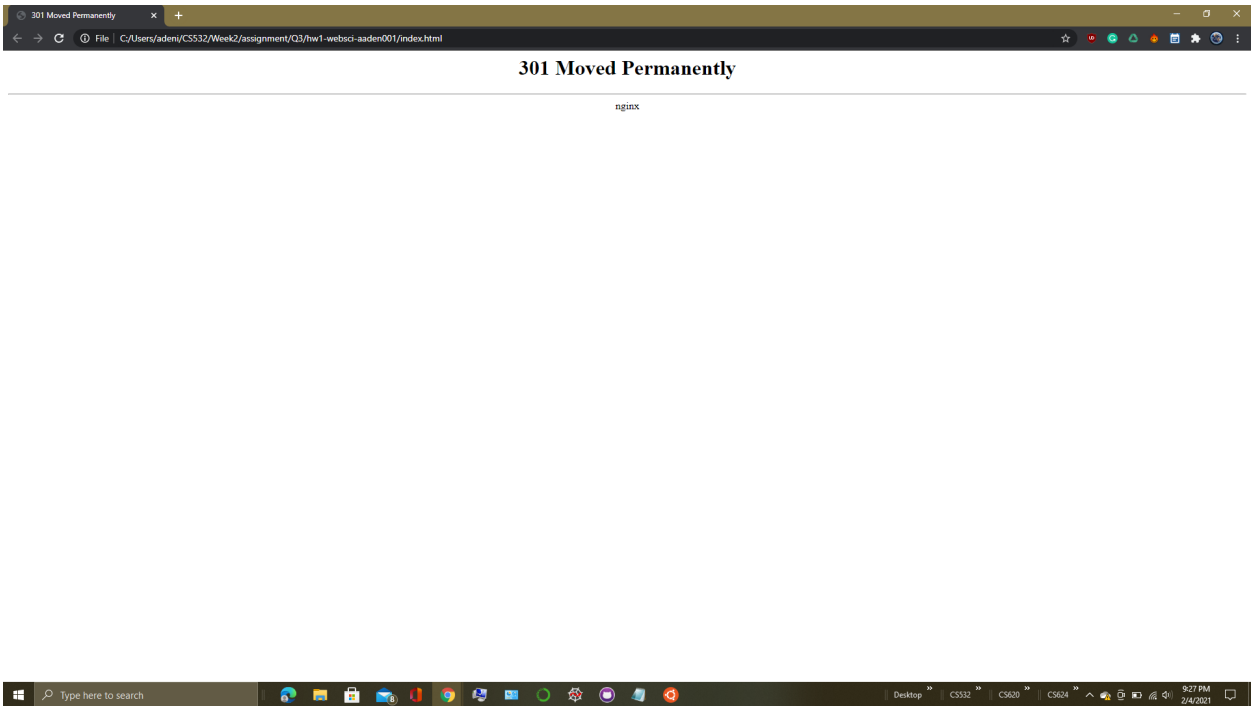


**Figure 3:** Question 2(c) output file in a the web browser

## Discussion

*The input command used to write the output to a .html format to so it can be viewed in the browser.*

```
curl -L http://www.cs.odu.edu/~weigle/courses/cs532/ua_echo.php --next
   -A "CS432/532" http://www.cs.odu.edu/~mweigle/cs532/ua_echo.php -o
   index.html
```

*I used -o to write to a file and gave the file name to be created and written to*

# Q3

*Write a Python program to find links to PDFs in a webpage.*
*Your program must do the following:*

- take the URI of a webpage as a command-line argument
- extract all the links from the page
- for each link, request the URI and use the Content-Type HTTP response header to determine if the link references a PDF file
- for all links that reference a PDF file, print the original URI (found in the source of the original HTML), the final URI (after any redirects), and the number of bytes in the PDF file. (Hint: Content-Length HTTP response header)

## Answer

```
# -*- coding: utf-8 -*-
"""
Created on Tue Feb  2 19:14:41 2021

@author: adeni
"""
#!/usr/bin/python3
import sys  # access command line aruguements
from bs4 import BeautifulSoup # html/xml parser
import requests #requests
from requests.exceptions import HTTPError # for error handling
from urllib.parse import urljoin #construct full url path

listOfargv = sys.argv
```

```
15 def contentTypeRequest(value):
16     try:
17         req = requests.get(value)
18         if req:
19             # check link content type
20             if( req.headers['Content-Type'] == "application/pdf"):
21                 print("This is a pdf file \n")
22                 print("URI: {} \nFinal URL: {} \nContent Length: {}
    bytes \n\n" .format(value, req.url,req.headers['content-length']))
23             else:
24                 print("Is Not a pdf File \n" )
25         else:
26             #displays status code error
27             print("Request Failed Status code {}\nPlease Enter a valid
    webpage link \n" .format(req.status_code))
28     except HTTPError as http_err:
29         print(f'HTTP error occurred: {http_err}')
30     except Exception as err:
31         print(f'Other error occurred: {err}')
32 
33 def prettyPrint(dictionary):
34     i = 1
35     for x in dictionary.keys():
36 
37         print("{}: {} ".format(i,x))
38         contentTypeRequest(x)
39         i = i+1
40 
41 def getLinksFromWebsite(base_URL):
42 
43     try:
44         # Get the request of the site
45         response = requests.get(base_URL)
46         blank_dict ={}
47         if response:
48             # removes bs4 warning
49             soup = BeautifulSoup(response.text, "html.parser" )
50             print("\nSite: {}" .format(base_URL) )
51             # gets all a tags that has href values
52             href_tags = soup.find_all('a',href=True)
53             #indirect way to extract href
54             for link in href_tags:
55                 if (blank_dict.get(urljoin(base_URL, link.get('href')))
    ) != None:
56                     # avoid duplicates
57                     pass
58                 else:
```

```
                    #Store link
                    blank_dict[urljoin(base_URL, link.get('href'))] =1
            #print dictionary
            prettyPrint(blank_dict)

        else:
            #displays status code error
            print("Request Failed Status code {}\nPlease Enter a valid
    webpage" .format(response.status_code))
    except HTTPError as http_err:
        print(f'HTTP error occurred: {http_err}')
    except Exception as err:
        print(f'Other error occurred: {err}')

if len(listOfargv) > 1:
    for x in range(1,len(listOfargv)):  # Run multiple times
        getLinksFromWebsite(listOfargv[x])
        print("\n")
else:
    print("Pass in a link")
```

**Listing 1:** Python sample code loaded from file

```
Site: https://www.cs.odu.edu/~mweigle/courses/cs532/pdfs.html
1: http://twitter.com/webscidl
Request Failed Status code 400
Please Enter a valid webpage link

2: https://doi.org/10.1108/IDD-05-2018-0014
Is Not a pdf File

3: http://www.cs.odu.edu/~mln/pubs/ht-2018/hypertext-2018-nwala-bootstrapping.pdf
This is a pdf file

URI: http://www.cs.odu.edu/~mln/pubs/ht-2018/hypertext-2018-nwala-bootstrapping.pdf
Final URL: https://www.cs.odu.edu/~mln/pubs/ht-2018/hypertext-2018-nwala-bootstrapping.pdf
Content Length: 994153bytes


4: http://www.cs.odu.edu/~mln/pubs/ipres-2018/ipres-2018-atkins-news-similarity.pdf
This is a pdf file

URI: http://www.cs.odu.edu/~mln/pubs/ipres-2018/ipres-2018-atkins-news-similarity.pdf
Final URL: https://www.cs.odu.edu/~mln/pubs/ipres-2018/ipres-2018-atkins-news-similarity.pdf
Content Length: 18995885bytes


5: https://arxiv.org/abs/1806.09082
Is Not a pdf File

6: http://www.cs.odu.edu/~mln/pubs/ipres-2018/ipres-2018-jones-off-topic.pdf
This is a pdf file

URI: http://www.cs.odu.edu/~mln/pubs/ipres-2018/ipres-2018-jones-off-topic.pdf
Final URL: https://www.cs.odu.edu/~mln/pubs/ipres-2018/ipres-2018-jones-off-topic.pdf
Content Length: 3119205bytes


7: https://arxiv.org/abs/1806.06870
Is Not a pdf File

8: http://www.cs.odu.edu/~mln/pubs/ipres-2018/ipres-2018-jones-archiveit.pdf
This is a pdf file
```

**Figure 4:** Sample Output display

## Discussion

*(1) This allows you to run as many arguments supplied to the program as a pretty output display*

```
72 if len(listOfargv) > 1:
73     for x in range(1,len(listOfargv)):  # Run multiple times
74         getLinksFromWebsite(listOfargv[x])
75         print("\n")
76 else:
77     print("Pass in a link")
```

**Listing 2:** The drive for the code:

*(2) Getting necessary libraries that will be used for the task*

```
8 import sys  # access command line aruguements
9 from bs4 import BeautifulSoup # html/xml parser
10 import requests #requests
11 from requests.exceptions import HTTPError # for error handling
12 from urllib.parse import urljoin #construct full url path
```

**Listing 3:** Python libraries used

*(3) Request and Beautiful soup libraries. Line 41 function,*

```
41 def getLinksFromWebsite(base_URL):

45         response = requests.get(base_URL)

49             soup = BeautifulSoup(response.text, "html.parser" )

52             href_tags = soup.find_all('a',href=True)
```

*takes the url supplied from the command line interface. It makes a get request using the request library. The request is then parsed as a text along with "html parser" as parameters. With this result in soup variable we can get all href values in each html a tags.*

*(4) Constructing Absolute path, avoiding duplicates and storing*

```
55                 if (blank_dict.get(urljoin(base_URL, link.get('href'))) != None:
56                     # avoid duplicates
57                     pass
58                 else:
59                     #Store link
60                     blank_dict[urljoin(base_URL, link.get('href'))] =1
```

*Line 55, using urljoin module from urlllib.parse library. We can be able to construct an absolute path url using the URI supplied from the command line and the href_tags.The href_tags contains some URI that are relative. This would convert all absolute URI to absolute path URI and would not affect abosulte path URI in href_tags. Line 55 also ensures that repeated values are avoid. We do nothing as shown in line 57. In line 60 the value is store in a dictionary*

*(5) Content Type Request function . Line 15 function,*

```
15 def contentTypeRequest(value):

20             if( req.headers['Content-Type'] == "application/pdf"):

22                 print("URI: {} \nFinal URL: {} \nContent Length: {} bytes \n\n" .format(value, req.url,req.headers['content-length']))
```

*Takes a URL and checks to see if content-type matches application/pdf. If there is a match We display the supplied link, links after so many redirection if there is, and the size of the content type in bytes. This is done for every link supplied as can be seen in figure 4*

# References


- https://stackoverflow.com/questions/44001007/scrape-the-absolute-url-instead-of-a-relative-path-in-python
- https://www.askpython.com/python/dictionary/check-if-key-exists-in-a-python-dictionary
- https://requests.readthedocs.io/en/master/user/quickstart/#response-content
- https://csacademy.com/app/graph_editor/