CS532 Web Science: Assignment 1

Finished on January 28, 2016

Dr. Michael L. Nelson

Naina Sai Tipparti
ntippart@cs.odu.edu

Contents

Problem 1 **2**
Question 2
Answer 2

Problem 2 **7**
Question 7
Answer 7

Problem 3 **11**
Question 11
Answer 12

Problem 1

Question

Demonstrate that you know how to use "curl" well enough to correctly POST data to a form. Show that the HTML response that is returned is "correct". That is, the server should take the arguments you POSTed and build a response accordingly. Save the HTML response to a file and then view that file in a browser and take a screen shot.

Answer

The *cURL* command is a handy command-line utility for making HTTP requests. Using *curl* to stream data, it can be a very useful troubleshooting tool. It also allows us to assess "raw" streaming performance. The following command *(See Figure: 1)* will get the content of the URL and display it in the terminal.

Make requests with data:
I have created a simple php page *curl_posted.php*, which accepts two arguments:

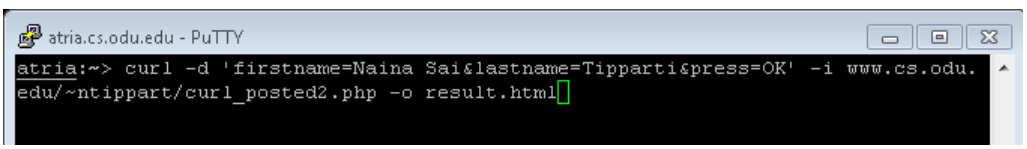


Figure 1: Screen shot of using curl Command

If there's a "normal" post, use -d to post. -d takes a full "post string", which is in the format

$$< variable1 >=< data1 > \& < variable2 >=< data2 > \&...$$

The "variable" names are the "names" set with "firstname=" in the $< input >$ tags, and the data is the contents you want to fill in for the inputs. The data **must** be properly URL encoded.

We can save the result of the curl command to a file by using -o/-O options.

- -o (lowercase o): result will be saved in the filename provided in the command line
- -O (uppercase O): filename in the URL will be taken and it will be used as the filename to store the result

To store the output in a file*(See Figure: 5)*, redirect it as shown above. This will also display some additional download statistics *(See Figure: 4.)*.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"><head>
 4 <meta http-equiv="Content-Type" content="text/html;
  charset=utf-8" />
<title>CS532 Web Science | Assignment 1</title>
</head>
<body>
<body>
<form method="POST" action="curl_posted2.php">
First name: <input type="text" name="firstname"><br>
Last name: <input type="text" name="lastname"><br>
<input type="submit" name="press" value="OK">
</form>
</body>
</html>
```

curl_posted.php

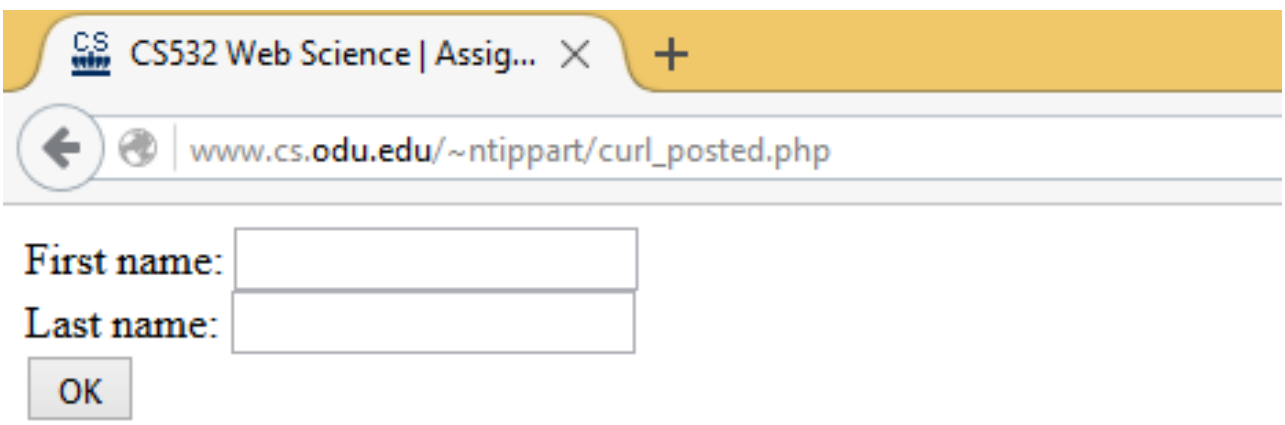


Figure 2: Screen shot of curl_posted.php page

Following php script is executed at the server side, when the form is submitted:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html;
 charset=utf-8" />
<title>CS532 Web Science | Assignment 1</title>
</head>
<body>
<?php
        $firstname = $_POST['firstname'];
        $lastname = $_POST['lastname'];
 echo '\n<h2> Your Full Name is '." ".$firstname."
".$lastname." ".'</h2>';
?>
</body>
</html>
```

curl_posted2.php

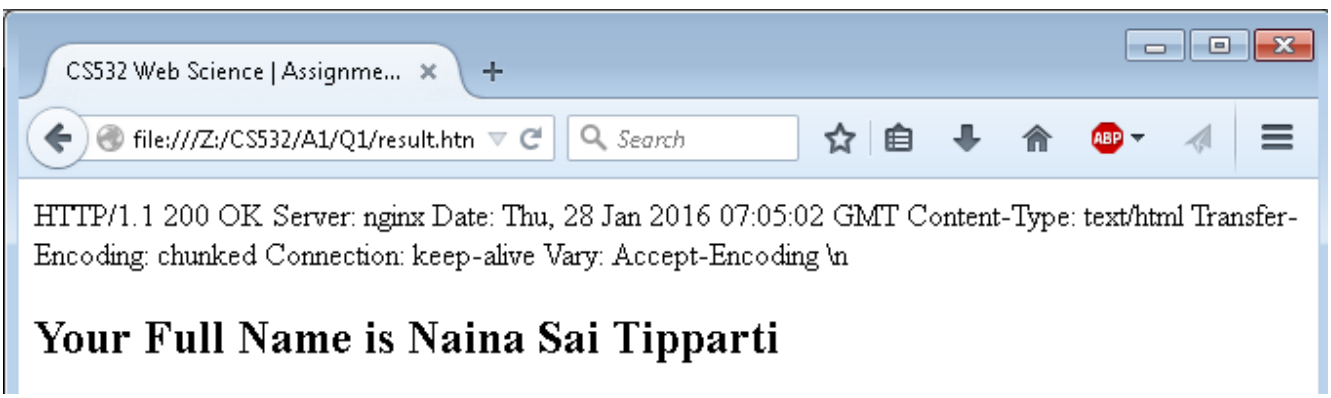
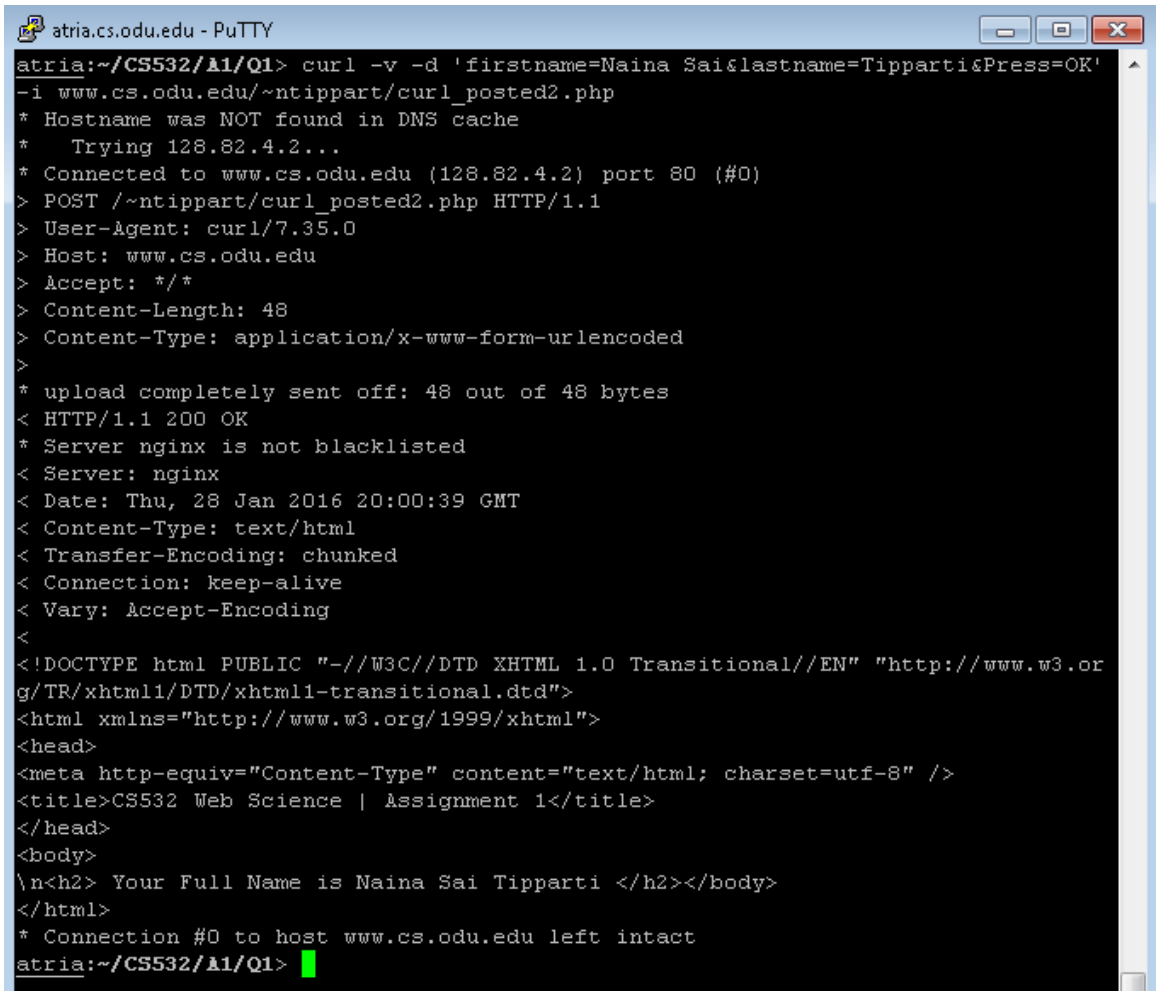


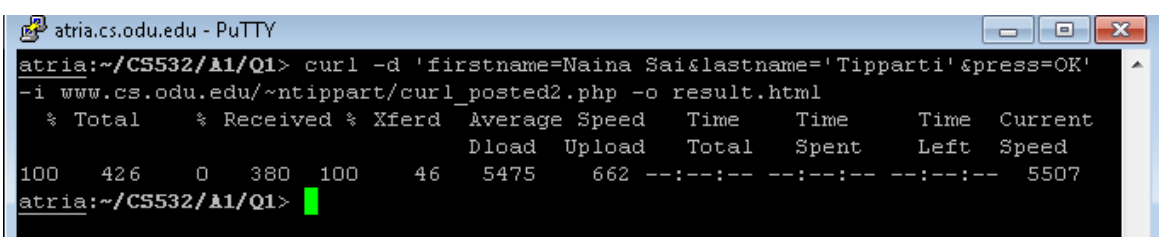
Figure 3: Screen shot of saved HTML response

If curl fails where it isn't supposed to, if the servers don't let you in, if you can't understand the responses: use the -v flag *(See Figure: 3)* to get verbose fetching. Curl will output lots of info and what it sends and receives in order to let the user see all client-server interaction (but it won't show you the actual data).



```
atria:~/CS532/A1/Q1> curl -v -d 'firstname=Naina Sai&lastname=Tipparti&Press=OK'
-i www.cs.odu.edu/~ntippart/curl_posted2.php
* Hostname was NOT found in DNS cache
*   Trying 128.82.4.2...
* Connected to www.cs.odu.edu (128.82.4.2) port 80 (#0)
> POST /~ntippart/curl_posted2.php HTTP/1.1
> User-Agent: curl/7.35.0
> Host: www.cs.odu.edu
> Accept: */*
> Content-Length: 48
> Content-Type: application/x-www-form-urlencoded
>
* upload completely sent off: 48 out of 48 bytes
< HTTP/1.1 200 OK
* Server nginx is not blacklisted
< Server: nginx
< Date: Thu, 28 Jan 2016 20:00:39 GMT
< Content-Type: text/html
< Transfer-Encoding: chunked
< Connection: keep-alive
< Vary: Accept-Encoding
<
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.or
g/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>CS532 Web Science | Assignment 1</title>
</head>
<body>
\n<h2> Your Full Name is Naina Sai Tipparti </h2></body>
</html>
* Connection #0 to host www.cs.odu.edu left intact
atria:~/CS532/A1/Q1>
```

Figure 4: Screen short of HTML Response



```
atria:~/CS532/A1/Q1> curl -d 'firstname=Naina Sai&lastname='Tipparti'&press=OK'
-i www.cs.odu.edu/~ntippart/curl_posted2.php -o result.html
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   426    0   380  100    46   5475    662 --:--:-- --:--:-- --:--:--  5507
atria:~/CS532/A1/Q1>
```

Figure 5: Screen shot of curl POST

Problem 2

Question

Write a Python program that:

1. takes as a command line argument a web page
2. extracts all the links from the page
3. lists all the links that result in PDF files, and prints out the bytes for each of the links. (note: be sure to follow all the redirects until the link terminates with a "200 OK".)
4. show that the program works on 3 different URIs, one of which needs to be: http://www.cs.odu.edu/~mln/teaching/cs532-s16/test/pdfs.html

Answer

In this python program, modules that are being used:

1. Beautiful Soup is an HTML/XML parser for Python that can turn even invalid markup into a parse tree. It provides simple, idiomatic ways of navigating, searching, and modifying the parse tree.

2. Validators can be any callable that takes a single parameter which checks the new value before it is assigned to the attribute. Validators are permitted to modify a received value so that it is appropriate for the attribute definition. For example, using int as a validator will cast a correctly formatted string to a number, or raise an exception if it can not. However. the correct way to use a validator that ensure the correct type is to use the Type validator.

3. Requests takes all of the work out of Python HTTP/1.1 making your integration with web services seamless. Theres no need to manually add query strings to your URLs, or to form-encode your POST data. Keep-alive and HTTP connection pooling are 100% automatic, powered by urllib3, which is embedded within Requests.

```
Veena Talapaneni@Veena ~
$ python q2.py http://www.cs.odu.edu/~mln/teaching/cs532-s16/test/pdfs.html

Extracting all pdf links from: http://www.cs.odu.edu/~mln/teaching/cs532-s16/test/pdfs.html

List of all PDFs Links:
http://bit.ly/1ZDatNK, File Size: 720476 bytes

http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-mink.pdf, File Size: 1254605 bytes

http://arxiv.org/pdf/1512.06195, File Size: 1748961 bytes

http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-off-topic.pdf, File Size: 4308768 bytes

http://www.cs.odu.edu/~mln/pubs/ht-2015/hypertext-2015-temporal-violations.pdf, File Size: 2184076 bytes

http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-stories.pdf, File Size: 1274604 bytes

http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-dictionary.pdf, File Size: 2350603 bytes

http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-annotations.pdf, File Size: 622981 bytes

http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-arabic-sites.pdf, File Size: 709420 bytes

http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-profiling.pdf, File Size: 639001 bytes

Veena Talapaneni@Veena ~
$
```

Figure 6: Output of q2.py at http://www.cs.odu.edu/~mln/teaching/cs532-s16/test/pdfs.html

```
Veena Talapaneni@Veena ~
$ python q2.py https://ws-dl.cs.odu.edu/Main/Pubs

Extracting all pdf links from: https://ws-dl.cs.odu.edu/Main/Pubs

List of all PDFs Links:
http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-mink.pdf, File Size: 1254605 bytes

http://www.cs.odu.edu/~mln/pubs/jcdl-2014/jcdl-2014-kelly-mink.pdf, File Size: 556863 bytes

http://www.cs.odu.edu/~mweigle/papers/padia-jcdl12.pdf, File Size: 755860 bytes

http://www.cs.odu.edu/~mkelly/papers/2013_ieeevis_boxofficeprediction.pdf, File Size: 122738 bytes

http://www.cs.odu.edu/~mweigle/papers/kelly-jcdl12.pdf, File Size: 152402 bytes

http://www.cs.odu.edu/~mkelly/papers/2014_dl_acid.pdf, File Size: 541843 bytes

http://www.cs.odu.edu/~mkelly/posters/2013_vis_boxoffice.pdf, File Size: 373897 bytes

http://www.cs.odu.edu/~mweigle/papers/aturban-tpdl15.pdf, File Size: 622981 bytes

http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-arabic-sites.pdf, File Size: 709420 bytes

http://www.cs.odu.edu/~mkelly/posters/2014_digpres_thumbnails.pdf, File Size: 100187772 bytes

http://www.cs.odu.edu/~mweigle/papers/ainsworth-jcdl11.pdf, File Size: 918103 bytes

http://www.cs.odu.edu/~mln/pubs/jcdl-2014/jcdl-2014-brunelle-damage.pdf, File Size: 2205546 bytes

Veena Talapaneni@Veena ~
$
```

Figure 7: Output of q2.py at https://ws-dl.cs.odu.edu/Main/Pubs

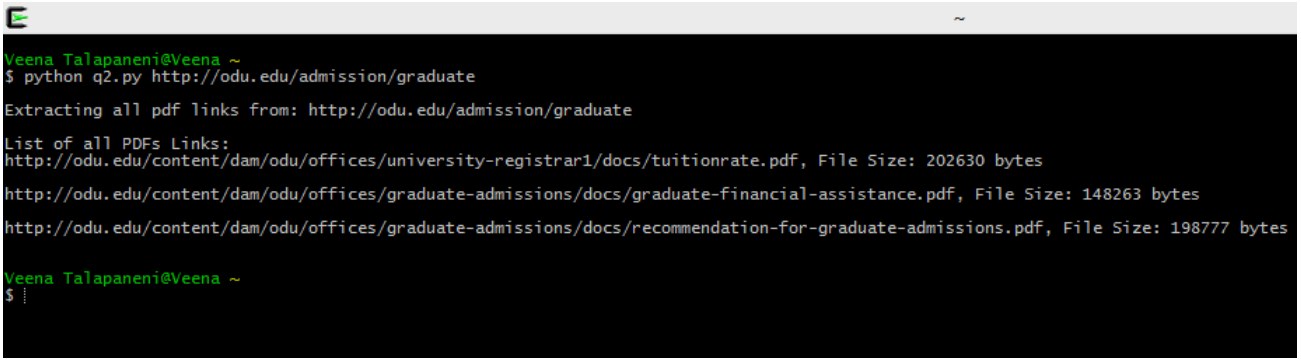


Figure 8: Output of q2.py at http://odu.edu/admission/graduate

Following python program *q2.py*, which accepts URL as argument and extracts PDF's from from the link:

```
import sys
import requests
import validators
import locale
from urllib.parse import urlparse
from bs4 import BeautifulSoup

def main(url):
    print('\nExtracting all pdf links from: %s' % url)

    if requests.get(url).status_code != 200:
        print('\nURL not Found!\n')
        return
    page = requests.get(url).text
    url = 'http://' + urlparse(url).netloc

    soup = BeautifulSoup(page, 'html.parser')
    all_links = []
    for link in soup.find_all('a'):
        urls = link.get('href')
        if ((len(urls) > 6 and urls[:7].lower() != 'http://')
        or len(urls) < 7) and urls[:8].lower() != 'https://':
            if urls[:2] == '//':
                urls = 'http:' + urls
            elif urls[0] != '/':
                urls = url + '/' + urls
            else:
                urls = url + urls
```

```

        try:
            r = requests.get(urls)
            if 'Content-Type' in r.headers and
            r.headers['Content-Type'] == 'application/pdf':
                if r.status_code == 200:
                    try:
                        all_links.append((urls,
                        r.headers['Content-Length']))
                    except KeyError:
                        r.headers['Content-Length'] = '???'
                        all_links.append((urls,
                        r.headers['Content-Length']))
        except requests.exceptions.SSLError:
            print('Couldn\'t open: %s.
            URL requires authentication.' % urls)
        except requests.exceptions.ConnectionError:
            print('Couldn\'t open: %s. Connection refused.' % urls
                )
    print('\nList of all PDFs Links:')
    pdf_links = set(all_links)
    all_links = list(pdf_links)
    if len(all_links) > 0:
        for i in range(len(pdf_links)):
            if all_links[i][1] == '???':
                print('%s, File Size: %s bytes \n'
                 % (all_links[i][0], all_links[i][1]))
            else:
                print('%s, File Size: %s bytes \n'
                % (all_links[i][0],
                locale.format("%d", int(all_links[i][1]),
                grouping=True)))
    else:
        print('\nNo PDFs links for above URI.')
    return
if __name__ == '__main__':
    if len(sys.argv) != 2:
        print('\nUsage: python q2.py [url]')
        sys.exit(-1)
    if not validators.url(sys.argv[1]):
        print('URL is Invalid, Please try again')
        sys.exit(1)
    main(sys.argv[1])
    sys.exit(0)
```

Problem 3

Question

Consider the "bow-tie" graph in the Broder et al. paper (fig 9):
http://www9.org/w9cdrom/160/160.html

Now consider the following graph:

A	→	B
B	→	C
C	→	D
C	→	A
C	→	G
E	→	F
G	→	C
G	→	H
I	→	H
I	→	J
I	→	K
J	→	D
L	→	D
M	→	A
M	→	N
N	→	D
O	→	A
P	→	G

For the above graph, give the values for:

IN:
SCC:
OUT:
Tendrils:
Tubes:
Disconnected:

Answer

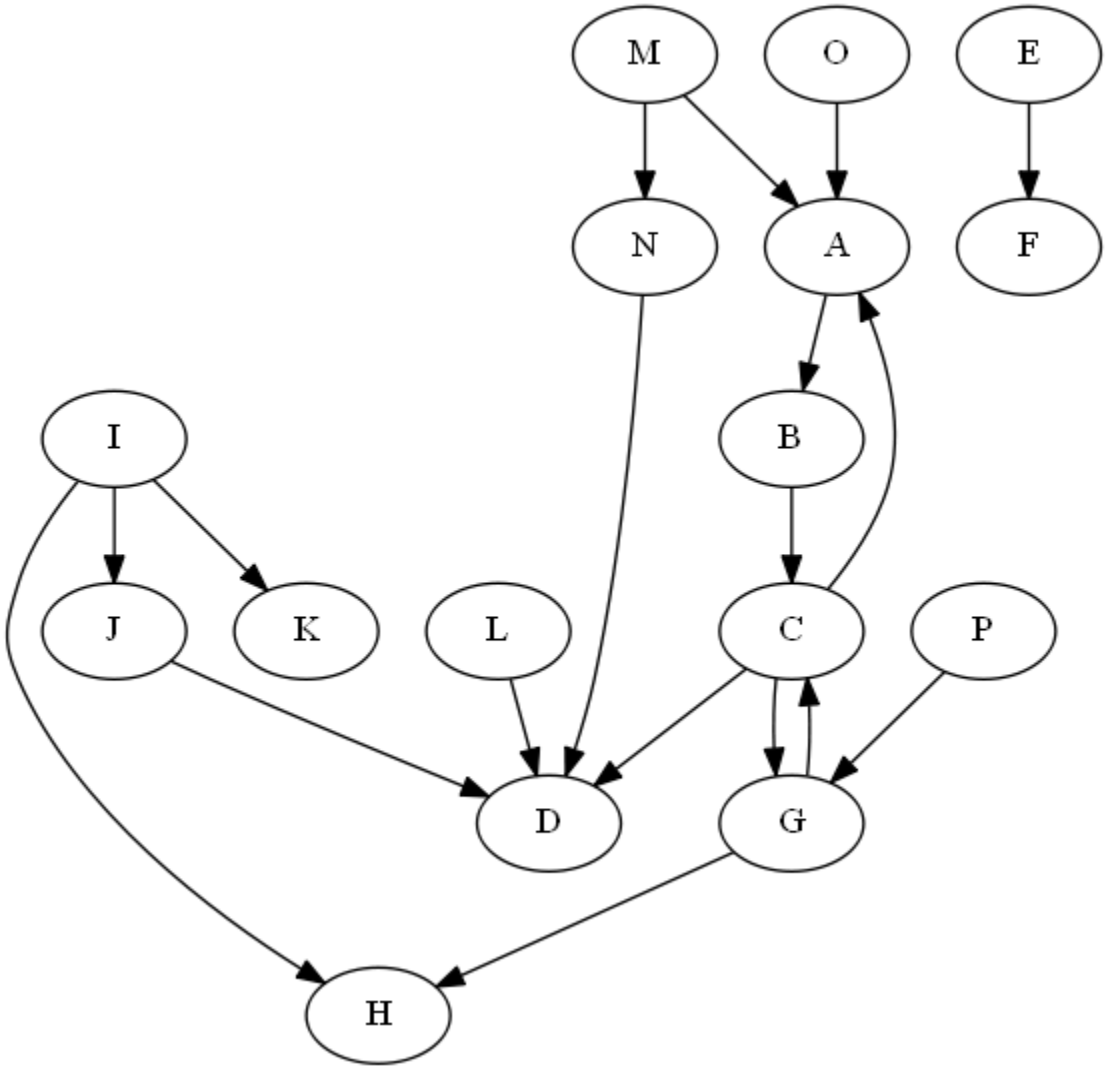


Figure 9: Drawing of example graph

Figure 9: is a graph of the included points, generated using Graphviz.

IN: {M,O,P}	"Connects into SCC, but not out from SCC"
SCC: {A,B,C,G}	"SCC – all contained nodes are interconnected"
OUT: {D,H}	"Connects out from SCC, but not in to SCC"
Tendrils: {I,J,K,L}	"In or out excluding all SCC"
Tubes: {N}	"IN → OUT or OUT → IN connection"
Disconnected:{E,F}	"Not connected to other sites"

The descriptions of each node is a matter of interpretation. Explanations have been provided for each value assignment.

IN: M, O, P

The IN components form the starting point for connection to the SCC[**?**]. They all sit at the start of the graph. In this graph, because of how the SCC and tubes are defined, the only IN components are M, O, P.

SCC: A, B, C, G

The Strongly Connected Component consists of those heavily linked items connected to from the list of nodes listed as part of IN[**?**]. It's an A, B, C, G world.

OUT: D, H

The OUT components exit the SCC, but do not link back to it[**?**]. The only members of OUT are D, H, who forms a sink from members of the SCC and the tendrils.

Tendrils: I, J, K, L

The *Tendrils* are pages that cannot reach the SCC or are not reached from the SCC[**?**]. The tendrils come from other graphs and only join the whole via $DorH$.

Tubes: N

Then there are *tubes*, which pass from IN to OUT without going through SCC[**?**]. N is a tube linking from M (from IN) to $DorH$ (from OUT).

Disconnected: E, F

The Disconnected components link to no one in the graph, and stand alone. They are not defined explicitly by Broder, et. al, but their meaning is implied within the paper.