Web Science: Assignment #1

Alexander Nwala

Mohd. Nauman Siddique

Thursday, January 31, 2019

Contents

Problem 1

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.
Feel free to use my simple server for sending POST requests: http://www.cs.odu.edu/~anwala/files/temp/namesEcho.php The server needs you to POST data for ”fname” and ”lname” fields.

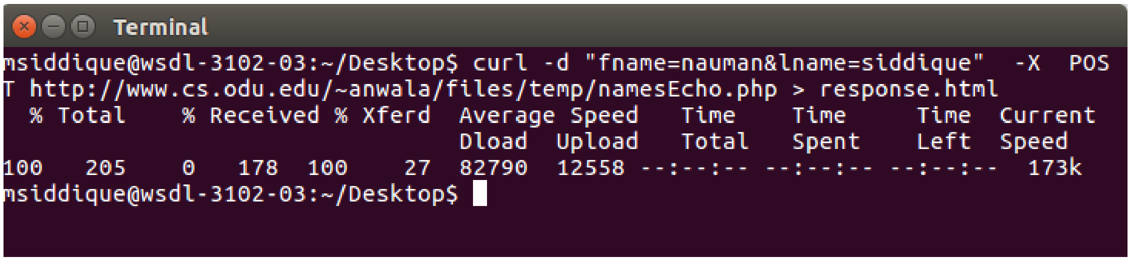
SOLUTION

The solution to the problem is as below:

1. **Curl Command**: curl [1] command to send parameters to a server and save its response to a html file is provided below.

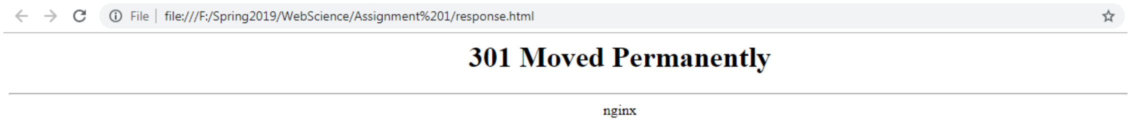
```
$ curl -d "fname=nauman&lname=siddique"  -X  POST http://www.cs.odu.edu/˜anwala/
    files/temp/namesEcho.php > response.html
```

2. **Curl Redirect**: The command redirects with 301 and its output can be seen in figure 1 and figure 2.



```
msiddique@wsdl-3102-03:~/Desktop$ curl -d "fname=nauman&lname=siddique"  -X  POS
T http://www.cs.odu.edu/~anwala/files/temp/namesEcho.php > response.html
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   205    0   178  100    27  82790  12558 --:--:-- --:--:-- --:--:--  173k
msiddique@wsdl-3102-03:~/Desktop$
```

Figure 1: Terminal screenshot for curl redirect on http://www.cs.odu.edu/~anwala/files/temp/namesEcho.php



file:///F:/Spring2019/WebScience/Assignment%201/response.html

301 Moved Permanently

nginx

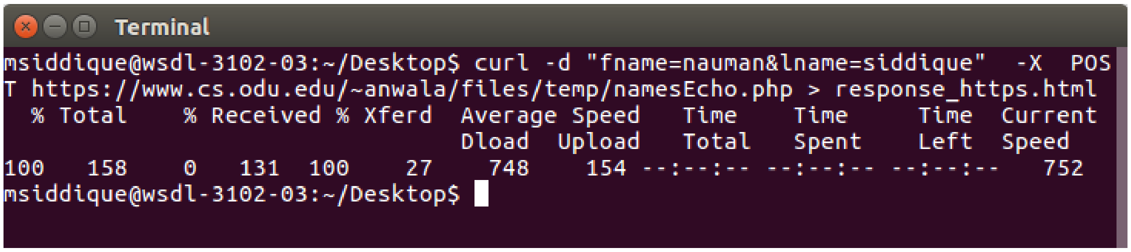
Figure 2: Browser screenshot for curl redirect on http://www.cs.odu.edu/~anwala/files/temp/namesEcho.php

3. **Curl https**: The command reolves with a status code of 200 and it can be seen in figure 3 and figure 4.

Problem 2

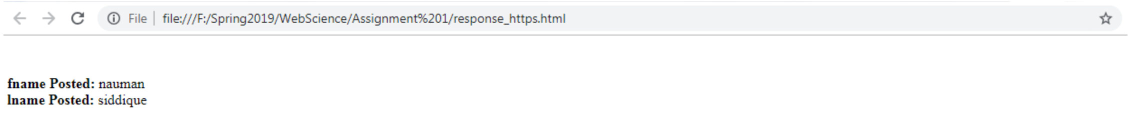
Write a Python program that:

Problem 2 continued on next page. . .



```
msiddique@wsdl-3102-03:~/Desktop$ curl -d "fname=nauman&lname=siddique"  -X  POS
T https://www.cs.odu.edu/~anwala/files/temp/namesEcho.php > response_https.html
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   158    0   131  100    27    748    154 --:--:-- --:--:-- --:--:--   752
msiddique@wsdl-3102-03:~/Desktop$ 
```

Figure 3: Terminal screenshot for curl on https://www.cs.odu.edu/~anwala/files/temp/namesEcho.php


File | file:///F:/Spring2019/WebScience/Assignment%201/response_https.html

fname Posted: nauman
lname Posted: siddique

Figure 4: Browser screenshot for curl on https://www.cs.odu.edu/~anwala/files/temp/namesEcho.php

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-s17/test/pdfs.html

SOLUTION

```
import requests
import bs4
import sys
import re

'''
Function to request pages
'''


def make_network_request(url):
    try:
        response = requests.get(url)
        if response.status_code == 200:
            if "application/pdf" in response.headers['content-type'] and url in
                list_extracted_links:
                file_pdf_links.write(url + "\n")
                print(response.text)
            if url not in list_extracted_links:
                extract_links(response.text)
        elif response.status_code == 301 or response.status_code == 302:
            for resp in response.history:
                list_extracted_links.append(resp.url)
    except requests.exceptions.MissingSchema as err:
        print("error: " + str(err) + "URL: " + url)
    except Exception as err:
        print("error: " + str(err))


'''
Function to extract links from Web Page
'''


def extract_links(content):
    soup = bs4.BeautifulSoup(content, "html.parser")
    link_a_tags = soup.find_all('a', href=True)
    for tags in link_a_tags:
        regex = re.compile(
            r'^(?:http|ftp)s?://'  # http:// or https://
            r'(?:(?:[A-Z0-9](?:[A-Z0-9-]{0,61}[A-Z0-9])?\.)+(?:[A-Z]{2,6}\.?|[A-Z0
                -9-]{2,}\.?)|'  # domain...
            r'localhost|'  # localhost...
            r'\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})'  # ...or ip
            r'(?::\d+)?'  # optional port
            r'(?:/?|[/?]\S+)$', re.IGNORECASE)
        if re.match(regex, tags["href"]) is not None:
            list_extracted_links.append(tags["href"])


'''
Function to check if queue is empty
'''


def check_list():
    while True:
        item = list_base_links[0]
        file_pdf_links.write("Links: " + item + "\n")
        file_pdf_links.write("PDF Links" + "\n")
        make_network_request(item)
        del list_base_links[0]
        if len(list_base_links) == 0:
            break
    while True:
        item = list_extracted_links[0]
        make_network_request(item)
        del list_extracted_links[0]
        if len(list_extracted_links) == 0:
            break

```

```
if __name__ == "__main__":
    list_base_links = []
    list_extracted_links = []
    file_pdf_links = open("PDF_Links.txt", "w")
    if len(sys.argv) > 1:
        argument_length = 0
        while argument_length < len(sys.argv):
            if argument_length > 0:
                list_base_links.append(sys.argv[argument_length])
                print("The arguments is: " + sys.argv[argument_length])
            argument_length += 1
    else:
        list_base_links.append("http://www.cs.odu.edu/~mln/teaching/cs532-s17/test/
            pdfs.html")
    check_list()
    file_pdf_links.close()
```

The solution for this problem is outlined by the following steps:

Assumption: For the purpose of this assignment, I have restricted extracting links from web pages to one hop from the argument url.
Command Line Argument: The code accepts command line argments. In case of no argument supplied the default argument is https://www.cs.odu.edu/~mln/teaching/cs532-s17/test/pdfs.html.
Extracting Links: For extracting links, Beautiful Soup library of python has been used to check for all the ”a” tags with href attribute.
Finding PDFs: For finding pdf links, we have relied on the content-type in the respponse header which should be ”application-pdf”.
Following Redirects: In case of redirects, the redirect links are added to the frontier which runs till the frontier runs itself empty.
Urls used for testing the code:

1. https://www.cs.odu.edu/~mln/teaching/cs532-s17/test/pdfs.html
2. https://odu.edu/compsci
3. https://www.cs.odu.edu/~mln/

Show Results: PDF links extracted from the three urls are shown below.

```
Links: http://www.cs.odu.edu/~mln/teaching/cs532-s17/test/pdfs.html
PDF Links
http://www.cs.odu.edu/~mln/pubs/ht-2015/hypertext-2015-temporal-violations.pdf
http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-annotations.pdf
http://arxiv.org/pdf/1512.06195
http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-off-topic.pdf
http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-stories.pdf
http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-profiling.pdf
http://www.cs.odu.edu/~mln/pubs/jcdl-2014/jcdl-2014-brunelle-damage.pdf
http://bit.ly/1ZDatNK
http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-mink.pdf
http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-arabic-sites.pdf
http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-dictionary.pdf
```

```
Links: https://odu.edu/compsci
```

```
PDF Links
https://www.cs.odu.edu/~advisor/advising/docs/18/Computer%20Science-Fall%202018.pdf
```

```
Links: https://www.cs.odu.edu/~mln/
PDF Links
http://www.cs.odu.edu/~mln/cv.pdf
http://www.cs.odu.edu/~mln/nsf-cv-2019.pdf
http://www.cs.odu.edu/~mln/mln-ad.pdf
```

Problem 3

Consider the "bow-tie" graph in the Broder et al. paper: http://snap.stanford.edu/class/cs224w-readings/broder00bowtie.pdf
Many have found this link useful: https://www.harding.edu/fmccown/classes/archive/comp475-s13/web-structure-homework.pdf Now consider the following graph:
$A - - > B$
$B - - > C$
$C - - > D$
$C - - > A$
$C - - > G$
$E - - > F$
$G - - > C$
$G - - > H$
$I - - > H$
$I - - > K$
$L - - > D$
$M - - > A$
$M - - > N$
$N - - > D$
$O - - > A$
$P - - > G$

For the above graph, give the values for:
IN:
SCC:
OUT:
Tendrils:
Tubes:
Disconnected:

SOLUTION
The graph for the bow-tie structure is converted as shown below and is being used as input to the python code.

```
0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0
1 0 0 1 0 0 1 0 0 0 0 0 0 0
```

Problem 3 continued on next page...

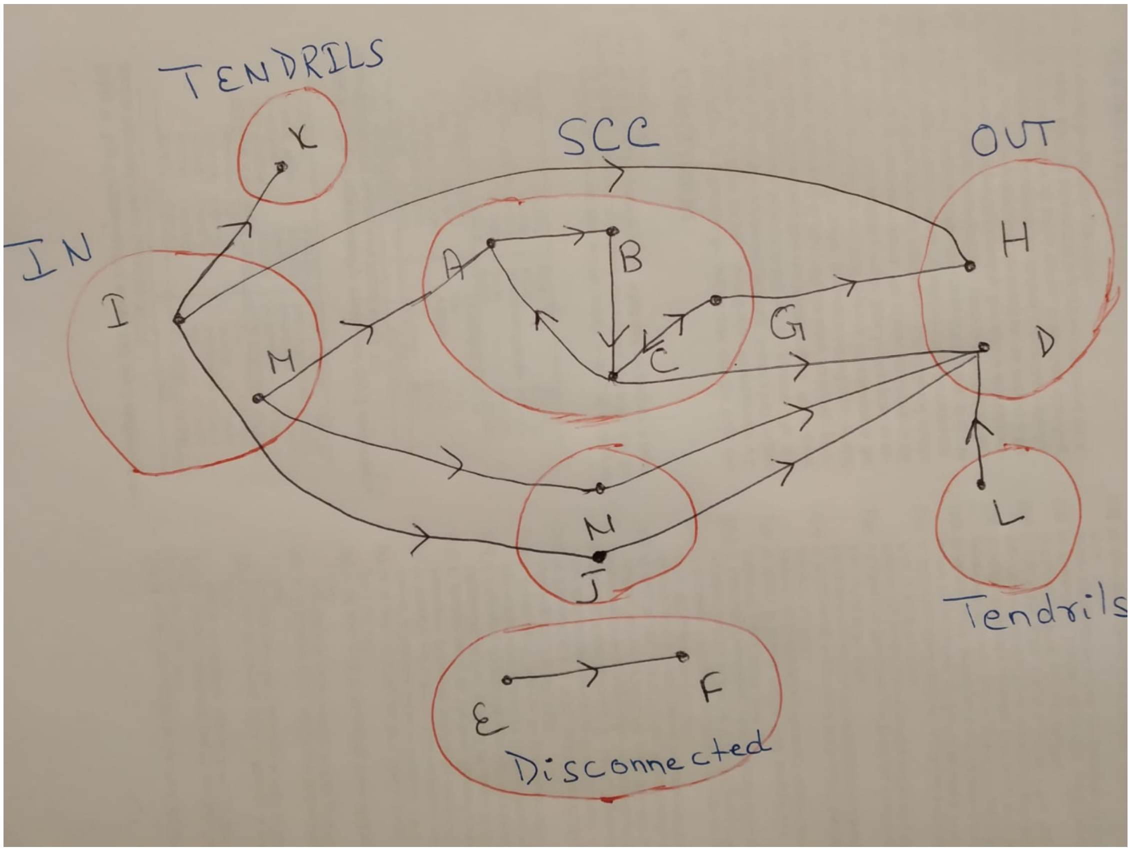


Figure 5: Graph for bow-tie

```
0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  1  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  1  0  0  0  0  1  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  1  1  0  0  0
0  0  0  1  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  1  0  0  0  0  0  0  0  0  0  0
1  0  0  0  0  0  0  0  0  0  0  0  0  1
0  0  0  1  0  0  0  0  0  0  0  0  0  0
```

```python
def bow_tie_structure():
    list_scc = []
    list_in = []
    list_out = []
    list_tendrils = []
    list_tubes = []
    list_disconnected = []
    outward_edges = []
    for row in range(0,len(graph)):
        outward_edges.append(False)
        # Check if in and out both entries present then SCC point
        for column in range(len(graph[0])):
```

Problem 3 continued on next page. . .

```
            if graph[row][column] == 1:
                outward_edges[row] = True
                for row_check in range(0, len(graph)):
                    if graph[row_check][row] == 1:
                        if label_points(row) not in list_scc:
                            list_scc.append(label_points(row))
                        break
    for row in range(0, len(graph)):
        if label_points(row) not in list_scc:
            if outward_edges[row]:
                if label_points(row) not in list_scc:
                    for column in range(len(graph[0])):
                        if graph[row][column] == 1:
                            # Check if point is IN
                            # Has only outgoing edges to SCC
                            if label_points(column) in list_scc:
                                if label_points(row) not in list_in:
                                    list_in.append(label_points(row))
            else:
                for row_check in range(0, len(graph)):
                    if graph[row_check][row] == 1:
                        # Check if point is OUT
                        # Has only incoming edges from SCC
                        if label_points(row_check) in list_scc:
                            if label_points(row) not in list_out:
                                list_out.append(label_points(row))

    for row in range(0, len(graph)):
        if label_points(row) not in list_scc and label_points(row) not in list_out and label_points(row) not in list_in:
            if outward_edges[row]:
                for column in range(len(graph[0])):
                    if graph[row][column] == 1:
                        # Check if point is Tendril
                        # Has only outgoing edges to OUT
                        if label_points(column) in list_out:
                            if label_points(row) not in list_tendrils:
                                list_tendrils.append(label_points(row))
                        else:
                            if label_points(row) not in list_disconnected:
                                list_disconnected.append(label_points(row))
            else:
                for column in range(len(graph[0])):
                    if graph[column][row] == 1:
                        # Check if point is Tendril
                        # Has only incoming edges to IN
                        if label_points(column) in list_in:
                            if label_points(row) not in list_tendrils:
                                list_tendrils.append(label_points(row))
                        else:
                            if label_points(row) not in list_disconnected:
                                list_disconnected.append(label_points(row))
    for points in list_scc:
        index_value = get_index(points)
        tube_point = True
        for column in range(0, len(graph[0])):
            if graph[index_value][column] == 1 and label_points(column) in list_scc:
                tube_point = False
            if tube_point:
                for row in range(0, len(graph)):
                    if graph[row][index_value] == 1 and label_points(row) in list_scc:
                        tube_point = False
            if tube_point:
                if label_points(index_value) not in list_tubes:
                    list_tubes.append(points)
    list_scc = list(set(list_scc) - set(list_tubes))
    print("SCC: " + str(list_scc))
    print("IN: " + str(list_in))
    print("OUT: " + str(list_out))
    print("Tendrils: " + str(list_tendrils))
    print("Tubes: " + str(list_tubes))
    print("Disconnected: " + str(list_disconnected))


def label_points(val):
    list_label = ["a", "b", "c","d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n"]
    return list_label[val]


def get_index(point):
    list_label = ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n"
        ]
    return list_label.index(point)
if __name__ == "__main__":
    file_graph = open("GraphInput.txt", "r")
    graph = []
    line_count = 0
    for line in file_graph:
        graph.append([])
        line_split = line.rstrip().split(" ")
        for entry in line_split:
            graph[line_count].append(int(entry))
        line_count += 1
    file_graph.close()
    bow_tie_structure()
```

The graph has been shown in figure 5 which shows all the different categories of points which have been explained below. The output is calculated with below mentioned categories of points:

1. **SCC**: points which have both incoming and outgoing edges. They are connected to points within IN, OUT and SCC.

2. **IN**: points which have only outgoing edges and are connected to SCC.

3. **OUT**: points which have only incoming edges and are connected to SCC.

4. **Tendrils**: points which have either incoming or outgoing edges connected to IN or OUT.

5. **Tube**: points which have inlinks from IN and outlinks to OUT.

6. **Disconnected**: points which are not connected to IN, OUT and SCC.

The output to the program listing all the points is shown in figure 6 and are also listed below.

1. **SCC**: a, b, c, g

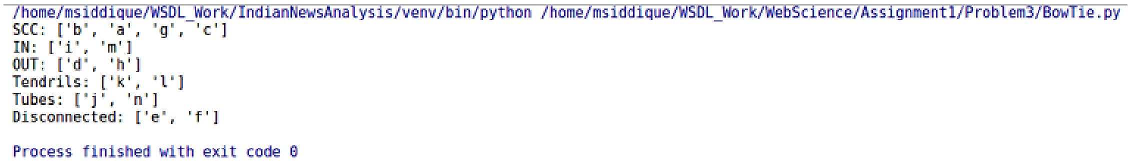
2. **IN**: i, m

3. **OUT**: d, h

4. **Tendrils**: k, l

5. **Tube**: j, n

6. **Disconnected**: e, f



```
/home/msiddique/WSDL_Work/IndianNewsAnalysis/venv/bin/python /home/msiddique/WSDL_Work/WebScience/Assignment1/Problem3/BowTie.py
SCC: ['b', 'a', 'g', 'c']
IN: ['i', 'm']
OUT: ['d', 'h']
Tendrils: ['k', 'l']
Tubes: ['j', 'n']
Disconnected: ['e', 'f']

Process finished with exit code 0
```

Figure 6: Output of BowTie Program

References

[1] Curl commands. https://gist.github.com. Accessed: 2019-01-30.