Web Science cs532-s16

Assignment 2 Report

By: Huan Huang

02/11/2016

Problem 1

Write a Python program that extracts 1000 unique links from Twitter. You might want to take a look at:

http://thomassileo.com/blog/2013/01/25/using-twitter-rest-api-v1-dot-1-with-python/

But there are many other similar resources available on the web. Note that only Twitter API 1.1 is currently available; version 1 code will no longer work.

Also note that you need to verify that the final target URI (i.e., the one that responds with a 200) is unique. You could have many different shortened URIs for www.cnn.com (t.co, bit.ly, goo.gl, etc.).

You might want to use the search feature to find URIs, or you can pull them from the feed of someone famous (e.g., Tim O'Reilly).

Hold on to this collection – we'll use it later throughout the semester.

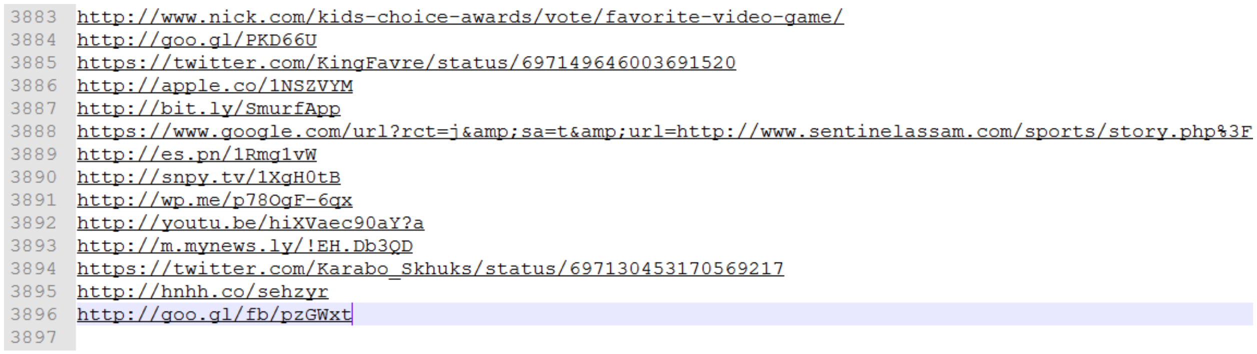
Answer

For this problem, I created 3 programs that handled the problem at each stage. But before I can start writing the programs, I had to register a new application with my twitter account. Once that is approved, I received my keys and tokens, with those keys and tokens I can access twitter's data through its API. So, the purpose of my first program is to use my keys and tokens to access twitter's data and listen for any tweets that contain the subject or phrase I set in my filter. originally, I intended to use Tweepy for this job, then, I came across the another module called Python Twitter Tools which made this part of assignment much simpler. When I setup the API object for tweet streaming, it automatically assign the different parts of the tweet into entities, all I have to do now is call for the entity urls to bring up the links in every tweet.

```
#!/usr/bin/python

from twitter import *

savefile = open('initialurls.txt','a')

# load my API credentials

accesstoken = "110800099-ZiXf6imxd24OVqw5nnm4nkHXmVwhX46QQI0Wb3Zt"
accesstokensecret = "FlDFz2Ol035FONaiL9uTzfqRauVArVwwVMuxiHDWQsmwo"
consumerkey = "1N9TkDHB9g4UIbjVj5j6idqnn"
consumersecret = "5Yjw1hRViWd5TloKLmEibPoregSltxO1bY7YJckjnecuSw4n9B"

# create twitter API object

```

```
auth = OAuth(accesstoken, accesstokensecret, consumerkey, consumersecret)
stream = TwitterStream(auth = auth, secure = True)

# set filter credentials
tweet_iter = stream.statuses.filter(track='sports, game')

for tweet in tweet_iter:

# print URLs found inside
  try:
    for url in tweet["entities"]["urls"]:

        print "%s" % url["expanded_url"]
        savefile.write("%s" % url["expanded_url"])
        savefile.write('\n')


  except (KeyError, UnicodeEncodeError) as e:
    pass
```

I let the program ran for about 30 minutes and collected 3896 links, which I saved in a text file called initialurls.txt(every file I mention will be provided in github).



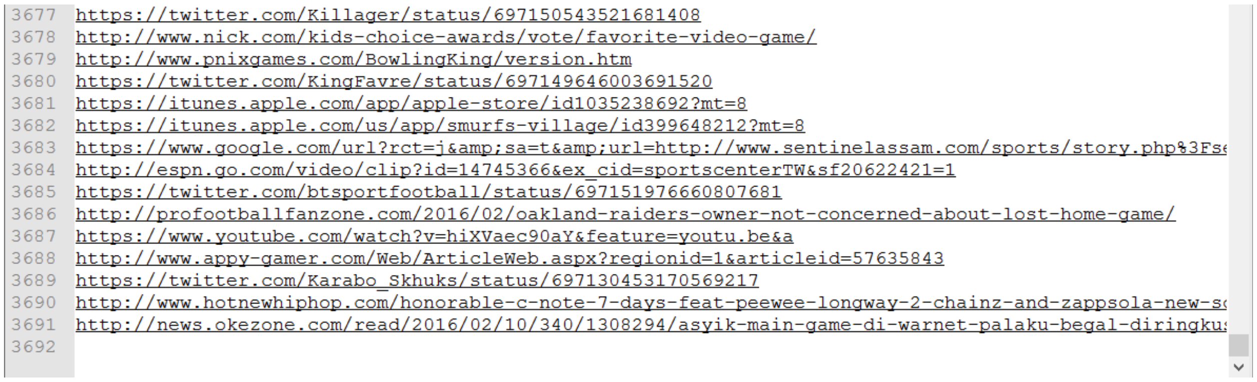
```
http://www.nick.com/kids-choice-awards/vote/favorite-video-game/
http://goo.gl/PKD66U
https://twitter.com/KingFavre/status/697149646003691520
http://apple.co/1NSZVYM
http://bit.ly/SmurfApp
https://www.google.com/url?rct=j&amp;sa=t&amp;url=http://www.sentinelassam.com/sports/story.php%3F
http://es.pn/1Rmg1vW
http://snpy.tv/1XgH0tB
http://wp.me/p78OgF-6qx
http://youtu.be/hiXVaec90aY?a
http://m.mynews.ly/!EH.Db3QD
https://twitter.com/Karabo_Skhuks/status/697130453170569217
http://hnhh.co/sehzyr
http://goo.gl/fb/pzGWxt
```

Figure 1: Sample of part of the initialurls.txt

Many of these URLs are short links or redirects, to handle that I wrote another program called urlredirect. I actually have two version of this program that does almost the same thing. The other version of this program uses request.get instead of HTTPRedirectHandler. But it operates much slower and sometimes stops at a URL without giving me any error or return control back to me nor was it in an infinite loop. It just stops there and this issue only happens sometimes. I could not figure out why and, therefore, given up on it.

```
import urllib2
import errno
import socket
import httplib
import ssl
import sys
import requests
from requests.exceptions import HTTPError

gett = open('initialurls.txt', 'r')
savefile = open('urlsRound2.txt', 'a')

s = socket.socket()
s.settimeout(5)

def getHeaders(urls):
  try:
      #conn = urllib2.urlopen(urls)
      opener = urllib2.build_opener(urllib2.HTTPRedirectHandler)
      request = opener.open(urls)

      print request.url
      savefile.write(request.url)
      savefile.write('\n')
      #print request.code

  except urllib2.HTTPError, x:
    print 'Ignoring', x.code
  except urllib2.URLError, e:
    print 'Ignoring', e.args
  except ssl.CertificateError , s:
    print 'Ignoring', s
  except socket.error , s:
    print 'Ignoring %s' % s
  except httplib.BadStatusLine , b:
    pass

for links in gett:
    urls = links
    getHeaders(urls);
```

The result is saved in a file called urlsRound2.txt.



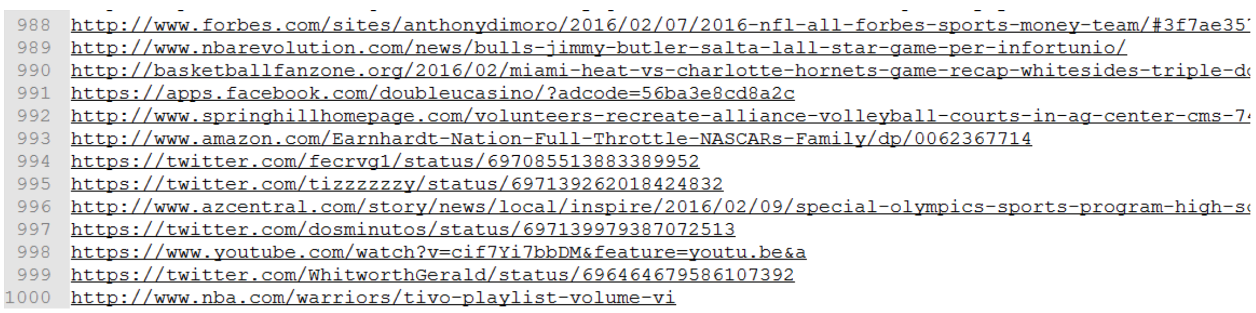
```
https://twitter.com/Killager/status/697150543521681408
http://www.nick.com/kids-choice-awards/vote/favorite-video-game/
http://www.pnixgames.com/BowlingKing/version.htm
https://twitter.com/KingFavre/status/697149646003691520
https://itunes.apple.com/app/apple-store/id1035238692?mt=8
https://itunes.apple.com/us/app/smurfs-village/id399648212?mt=8
https://www.google.com/url?rct=j&amp;sa=t&amp;url=http://www.sentinelassam.com/sports/story.php%3Fs
http://espn.go.com/video/clip?id=14745366&ex_cid=sportscenterTW&sf20622421=1
https://twitter.com/btsportfootball/status/697151976660807681
http://profootballfanzone.com/2016/02/oakland-raiders-owner-not-concerned-about-lost-home-game/
https://www.youtube.com/watch?v=hiXVaec90aY&feature=youtu.be&a
http://www.appy-gamer.com/Web/ArticleWeb.aspx?regionid=1&articleid=57635843
https://twitter.com/Karabo_Skhuks/status/697130453170569217
http://www.hotnewhiphop.com/honorable-c-note-7-days-feat-peewee-longway-2-chainz-and-zappsola-new-s
http://news.okezone.com/read/2016/02/10/340/1308294/asyik-main-game-di-warnet-palaku-begal-diringku
```

Figure 2: Sample of Part of urlround2.txt

The links I have now are all final links that would return a 200 response code, but there are too many duplicates and I only need 1000 unique links. Therefore, I created another program called uniqueurls.py to give me 1000 unique links. It just use *set* function to remove duplicate strings and remove all lines after line 1000.

```python

gett = open('urlsRound2.txt', 'r')
savefile = open('unqiueurls.txt', 'a')
savefile2 = open('requiredurls.txt', 'a')

#set function remove duplicates in a list
unqiue = set(gett)
for link in unqiue:
  #print link
  savefile.write(link)

#get the top 1000 strings
with open('unqiueurls.txt', 'r') as unqiueurls:
  top1000 = [next(unqiueurls) for x in xrange(1000)]
for links in top1000:
  savefile2.write(links)


#i = 0

#def forloops(unqiue, i):
  #for link in unqiue:
    #i = i + 1
    #print link
    #savefile.write(link)
    #return i


#while i < 1000:
  #i = forloops(unqiue, i)
```

The final result of 1000 unique URLs are saved in a file called requiredurls.txt.



```
http://www.forbes.com/sites/anthonydimoro/2016/02/07/2016-nfl-all-forbes-sports-money-team/#3f7ae35
http://www.nbarevolution.com/news/bulls-jimmy-butler-salta-lall-star-game-per-infortunio/
http://basketballfanzone.org/2016/02/miami-heat-vs-charlotte-hornets-game-recap-whitesides-triple-d
https://apps.facebook.com/doubleucasino/?adcode=56ba3e8cd8a2c
http://www.springhillhomepage.com/volunteers-recreate-alliance-volleyball-courts-in-ag-center-cms-7
http://www.amazon.com/Earnhardt-Nation-Full-Throttle-NASCARs-Family/dp/0062367714
https://twitter.com/fecrvg1/status/697085513883389952
https://twitter.com/tizzzzzzy/status/697139262018424832
http://www.azcentral.com/story/news/local/inspire/2016/02/09/special-olympics-sports-program-high-s
https://twitter.com/dosminutos/status/697139979387072513
https://www.youtube.com/watch?v=cif7Yi7bbDM&feature=youtu.be&a
https://twitter.com/WhitworthGerald/status/696464679586107392
http://www.nba.com/warriors/tivo-playlist-volume-vi
```

Figure 3: Sample of part of requiredurls.txt

Problem 2

Download the TimeMaps for each of the target URIs. We'll use the ODU Memento Aggregator, so for example:

URI-R = http://www.cs.odu.edu/

URI-T = http://mementoproxy.cs.odu.edu/aggr/timemap/link/1/http://www.cs.odu.edu/

Create a histogram* of URIs vs. number of Mementos (as computed from the TimeMaps). For example, 100 URIs with 0 Mementos, 300 URIs with 1 Memento, 400 URIs with 2 Mementos, etc.

* = https://en.wikipedia.org/wiki/Histogram

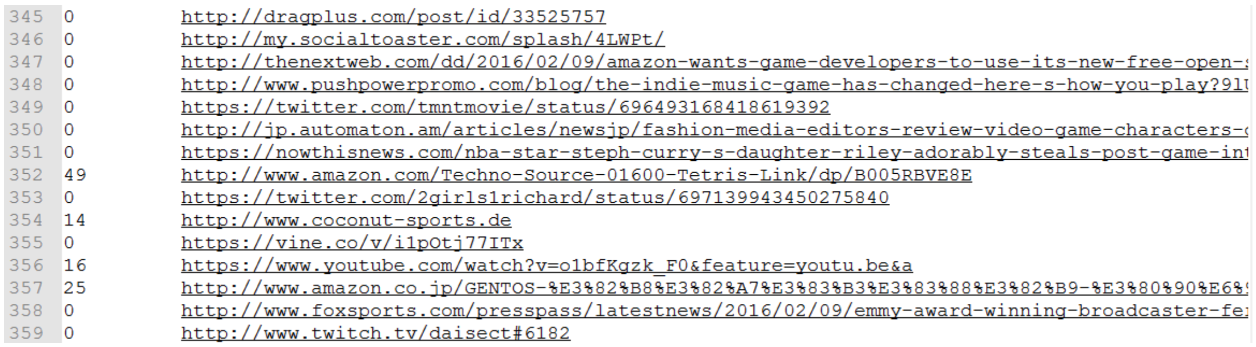
Answer

For this problem, my approach is this, for every link in the 1000 links, attach the url http://mementoproxy.cs.odu.edu/aggr/timemap/link/1/ in front of it. Then, use curl or urllib2.urlopen or whatever to open the newly generated links. (Be careful, if you are using curl, make sure to put the entire url inside "".)In the response, whenever the strings rel="memento", rel="first memento", rel="last memento", rel="memento first", rel="memento last" appear, it means there is a memento. Therefore, I just have to count them and add them up. Also, the response of some sites may contain the 'rel="timemap"', it means this site have more mementos in another link, make sure to open them up and count the mementos in there as well.

```python
import commands
import os, sys
import re
import requests
import urllib2

readfile = open('requiredurls.txt', 'r')
savefile = open('countMemento.txt', 'a')
savefile2 = open('forgraph.txt', 'a')
#inputUrlsList = readfile.readlines()
#readfile.close()

timeMap = re.compile(r'<[^>]+>;rel\w*?=\w*?"timemap".*?')


def nextpage(foundTimeMap):

  while len(foundTimeMap) == 1 :
    url = foundTimeMap[0]
    stripedURL = url.strip('<')
    urlStriped = stripedURL.split('>')
    nextPageLink = urlStriped[0]

    command = 'curl --silent "'+ nextPageLink + '"'
    response = os.popen(command).read()

    count = response.count('rel="memento"')
    count2 = response.count('rel="first memento"')
    count3 = response.count('rel="last memento"')
    count4 = response.count('rel="memento first"')
    count5 = response.count('rel="memento last"')
    countbig = count + count2 + count3 + count4 + count5
    foundTimeMap = timeMap.findall(response)

    if countbig != 0:
      return countbig

  return 0


for link in readfile:
  command = 'curl --silent "http://mementoproxy.cs.odu.edu/aggr/timemap/link
    /1/' + link.strip() + '"'
  response = os.popen(command).read()
  #command = 'http://mementoproxy.cs.odu.edu/aggr/timemap/link/1/' + link
  #response = urllib2.urlopen(url=command,timeout=10)
  #time_map = response.read()

  count = response.count('rel="memento"')
  count2 = response.count('rel="first memento"')
  count3 = response.count('rel="last memento"')
  count4 = response.count('rel="memento first"')
  count5 = response.count('rel="memento last"')

  foundTimeMap = timeMap.findall(response)
  returnedcount = nextpage(foundTimeMap)

```

```
finalcount = count + count2 + count3 + count4 + count5 + returnedcount

output = "{:<10}{}".format(finalcount , link)
print output

savefile.write(output)
savefile2.write(str(finalcount))
savefile2.write('\n')
```

The outputs are saved in two files, one contain the count and corresponding links(which is used later in problem 3), and another one only contain the count number(which is used for the graph).

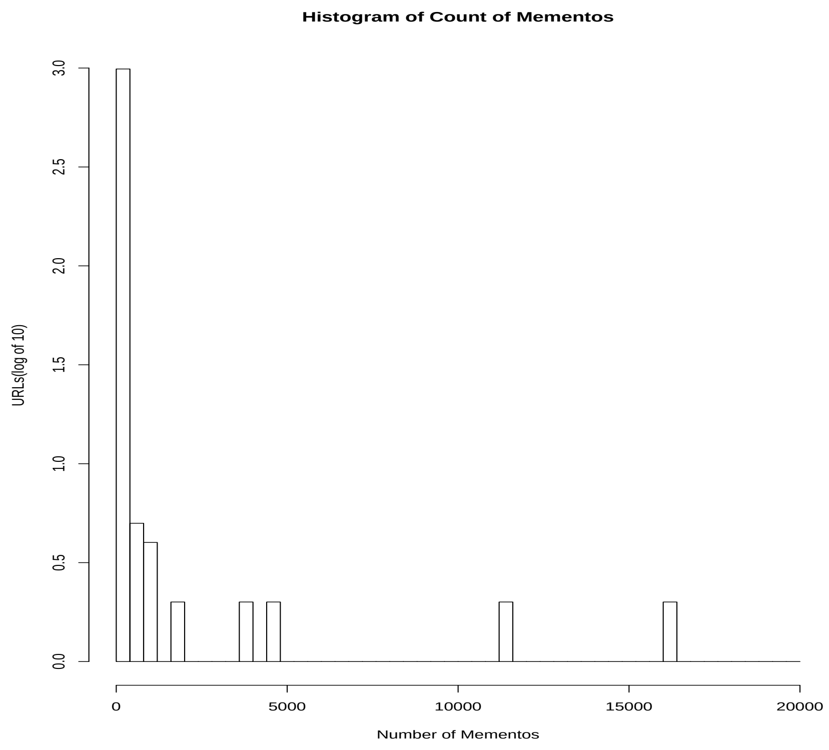


```
0         http://dragplus.com/post/id/33525757
0         http://my.socialtoaster.com/splash/4LWPt/
0         http://thenextweb.com/dd/2016/02/09/amazon-wants-game-developers-to-use-its-new-free-open-
0         http://www.pushpowerpromo.com/blog/the-indie-music-game-has-changed-here-s-how-you-play?91
0         https://twitter.com/tmntmovie/status/696493168418619392
0         http://jp.automaton.am/articles/newsjp/fashion-media-editors-review-video-game-characters-
0         https://nowthisnews.com/nba-star-steph-curry-s-daughter-riley-adorably-steals-post-game-in
49        http://www.amazon.com/Techno-Source-01600-Tetris-Link/dp/B005RBVE8E
0         https://twitter.com/2girls1richard/status/697139943450275840
14        http://www.coconut-sports.de
0         https://vine.co/v/i1pOtj77ITx
16        https://www.youtube.com/watch?v=o1bfKgzk_F0&feature=youtu.be&a
25        http://www.amazon.co.jp/GENTOS-%E3%82%B8%E3%82%A7%E3%83%B3%E3%83%88%E3%82%B9-%E3%80%90%E6%
0         http://www.foxsports.com/presspass/latestnews/2016/02/09/emmy-award-winning-broadcaster-fe
0         http://www.twitch.tv/daisect#6182
```

Figure 4: Sample of part of countMemento.txt

Below is the R code and graph of the histogram.

```
data2 = read.table(file.choose(), header = F)

r = hist(data2$V1, ylim = c(0, 1000), breaks = seq(from=0, to=20000, by=400),
         plot = F)

r$counts = log10(r$counts+1)


plot(r, main="Histogram of Count of Mementos", xlab = "Number of Mementos",
     ylab = "URLs(log of 10)")
```



Problem 3

Estimate the age of each of the 1000 URIs using the "Carbon Date" tool:

http://ws-dl.blogspot.com/2014/11/2014-11-14-carbon-dating-web-version-20.html

Note: you'll should download the library and run it locally; don't try to use the web service.

For URIs that have ¿ 0 Mementos and an estimated creation date, create a graph with age (in days) on one axis and number of mementos on the other.

Not all URIs will have Mementos, and not all URIs will have an estimated creation date. State how many fall into either categories.

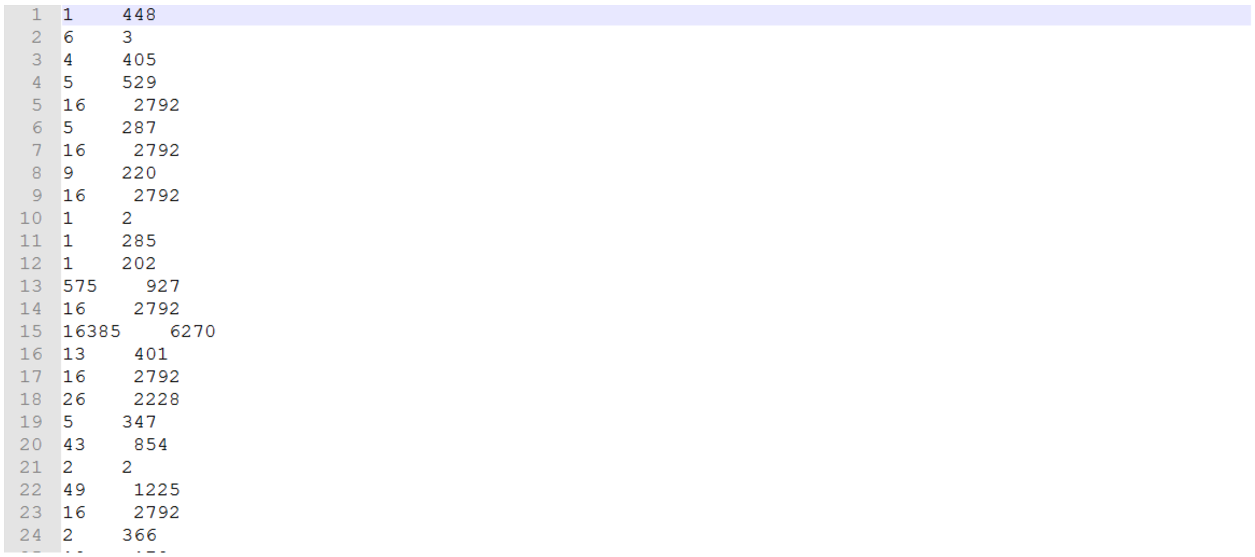
Answer

For this problem, we have to create a Bitly account first. While waiting for your verification mail to activate your account, go install PhantomJS, download CasperJS, and download CarbonDate tools. CasperJS is just a compressed folder with some files in it, just put it somewhere you can remember, but make sure you create a short link in /usr/local/bin that points to the casperjs file in the bin fold in the CasperJS folder. Now, activate your Bitly account and generate a token. Open the CarbonDate tool folder and open the config file, put your token in the right place and make sure the “CasperJSLocation” is set to where you created your short link towards the casperjs file. Now, we can finally start on the program itself.

To carbon date the 1000 links, I actually went into the local.py file and altered it. Since we are only looking for the estimated creation date, I just have the *cd* function return the variable *lowest*. Once I have the values outside the function, I convert it from string format back to object and use the time of now to subtract it to get the days since the site's creation. The result is saved in carbonDate.txt file.

```python
from checkForModules import checkForModules
import json
from ordereddict import OrderedDict
#import simplejson
import urlparse
import re

from getBitly import getBitlyCreationDate
from getArchives import getArchivesCreationDate
from getGoogle import getGoogleCreationDate
from getBacklinks import *
from getLowest import getLowest

from getLastModified import getLastModifiedDate
#Topsy service is no longer available
#from getTopsyScrapper import getTopsyCreationDate
from htmlMessages import *
from pprint import pprint

from threading import Thread
import Queue
import datetime

import os,sys, traceback

readfile = open('countMemento.txt', 'r')

savefile = open('carbonDate.txt', 'a')


def cd(url, backlinksFlag = False):

    #print 'Getting Creation dates for: ' + url


    #scheme missing?
    parsedUrl = urlparse.urlparse(url)
    if( len(parsedUrl.scheme)<1 ):
        url = 'http://'+url


    threads = []
    outputArray =['','','','','','']
    now0 = datetime.datetime.now()


    lastmodifiedThread = Thread(target=getLastModifiedDate, args=(url,
   outputArray, 0))
    bitlyThread = Thread(target=getBitlyCreationDate, args=(url, outputArray,
   1))
    googleThread = Thread(target=getGoogleCreationDate, args=(url, outputArray
   , 2))
    archivesThread = Thread(target=getArchivesCreationDate, args=(url,
   outputArray, 3))

    if( backlinksFlag ):
        backlinkThread = Thread(target=getBacklinksFirstAppearanceDates, args
   =(url, outputArray, 4))

    #topsyThread = Thread(target=getTopsyCreationDate, args=(url, outputArray,
     5))


    # Add threads to thread list
    threads.append(lastmodifiedThread)
    threads.append(bitlyThread)
    threads.append(googleThread)
    threads.append(archivesThread)

    if( backlinksFlag ):
        threads.append(backlinkThread)

    #threads.append(topsyThread)


    # Start new Threads
    lastmodifiedThread.start()
    bitlyThread.start()
    googleThread.start()
    archivesThread.start()

    if( backlinksFlag ):
        backlinkThread.start()

    #topsyThread.start()


    # Wait for all threads to complete
    for t in threads:
        t.join()

    # For threads
    lastmodified = outputArray[0]
    bitly = outputArray[1]
    google = outputArray[2]
    archives = outputArray[3]
```

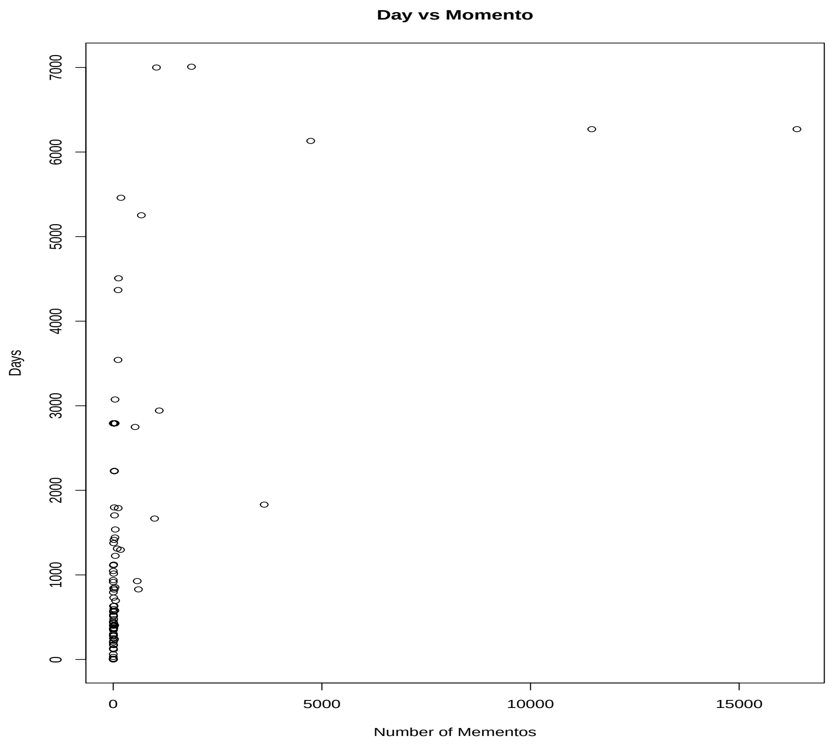
```python

    if( backlinksFlag ):
        backlink = outputArray[4]
    else:
        backlink = ''

    #topsy = outputArray[5]

    #note that archives["Earliest"] = archives[0][1]
    try:
        #lowest = getLowest([lastmodified, bitly, google, archives[0][1], backlink, topsy]) #for thread
        lowest = getLowest([lastmodified, bitly, google, archives[0][1], backlink]) #for thread
    except:
       print sys.exc_type, sys.exc_value , sys.exc_traceback



    result = []

    result.append(("URI", url))
    result.append(("Estimated Creation Date", lowest))
    result.append(("Last Modified", lastmodified))
    result.append(("Bitly.com", bitly))
    result.append(("Topsy.com", "Topsy is out of service"))
    result.append(("Backlinks", backlink))
    result.append(("Google.com", google))
    result.append(("Archives", archives))
    values = OrderedDict(result)

    #r = json.dumps(values, sort_keys=False, indent=2, separators=(',', ': '))

    now1 = datetime.datetime.now() - now0


    #print "runtime in seconds: "
    #print now1.seconds
    #print r
    #print 'runtime in seconds:  ' +  str(now1.seconds) + '\n' + r + '\n'
    #savefile.write(lowest)

    return lowest

nowdaate = datetime.datetime.now()

for line in readfile:
    link = line.split(" ")
    url = link[-1]
    returndate = cd(url)

    if returndate != "":
        date = datetime.datetime.strptime(returndate, '%Y-%m-%dT%H:%M:%S')
        dayys = (nowdaate - date).days

        print dayys
        print type(dayys)
        print link[0]
        print type(link[0])
        output = "{}    {}".format(link[0], dayys)
        savefile.write(output)
        savefile.write('\n')



#if len(sys.argv) == 1:
    #print "Usage: ", sys.argv[0] + " url backlinksOnOffFlag ( e.g: " + sys.argv[0] + " http://www.cs.odu.edu [--compute-backlinks] )"
#elif len(sys.argv) == 2:
    #fix for none-thread safe strptime
    #If time.strptime is used before starting the threads, then no exception is raised (the issue may thus come from strptime.py not being imported in a thread safe manner). -- http://bugs.python.org/issue7980
    #time.strptime("1995-01-01T12:00:00", '%Y-%m-%dT%H:%M:%S')
    #cd(sys.argv[1])
#elif len(sys.argv) == 3:
    #datimetime.strptime("1995-01-01T12:00:00", '%Y-%m-%dT%H:%M:%S')

    #if(sys.argv[2] == '--compute-backlinks'):
        #cd(sys.argv[1], True)
    #else:
        #cd(sys.argv[1])
```



```
1     448
6     3
4     405
5     529
16     2792
5     287
16     2792
9     220
16     2792
1     2
1     285
1     202
575     927
16     2792
16385     6270
13     401
16     2792
26     2228
5     347
43     854
2     2
49     1225
16     2792
2     366
```

Figure 5: Sample of part of carbonDate.txt

Now to finish the problem just use the data in carbonDate.txt to create a graph.


Day vs Momento
Days
Number of Mementos