Building a Search Engine - Introduction

This week we will download and run a simple version of the Google PageRank Algorithm. Here is an early paper by Larry Page and Sergy Brin, the founders of Google, that describes their early thoughts about the algorithm:

<http://infolab.stanford.edu/~backrub/google.html>

We will provide you with sample code and lectures that walk through the sample code:

<https://www.py4e.com/code3/pagerank.zip>

There is not a lot of new code to write - it is mostly looking at the code and making the code work. You will be able to spider some simple content that we provide and then play with the program to spider some other content. Part of the fun of this assignment is when things go wrong and you figure out how to solve a problem when the program wanders into some data that breaks its retrieval and parsing. So you will get used to starting over with a fresh database and running your web crawl.

Diagram

Description automatically generated

spider.py

**import** sqlite3  
**import** urllib.error  
**import** ssl  
**from** urllib.parse **import** urljoin  
**from** urllib.parse **import** urlparse  
**from** urllib.request **import** urlopen  
**from** bs4 **import** BeautifulSoup  
  
*# Ignore SSL certificate errors*ctx = ssl.create\_default\_context()  
ctx.check\_hostname = **False**ctx.verify\_mode = ssl.CERT\_NONE  
  
conn = sqlite3.connect(**'spider.sqlite'**)  
cur = conn.cursor()  
  
cur.execute(**'''CREATE TABLE IF NOT EXISTS Pages  
 (id INTEGER PRIMARY KEY, url TEXT UNIQUE, html TEXT,  
 error INTEGER, old\_rank REAL, new\_rank REAL)'''**)  
  
cur.execute(**'''CREATE TABLE IF NOT EXISTS Links  
 (from\_id INTEGER, to\_id INTEGER, UNIQUE(from\_id, to\_id))'''**)  
  
cur.execute(**'''CREATE TABLE IF NOT EXISTS Webs (url TEXT UNIQUE)'''**)  
  
*# Check to see if we are already in progress...*cur.execute(**'SELECT id,url FROM Pages WHERE html is NULL and error is NULL ORDER BY RANDOM() LIMIT 1'**)  
row = cur.fetchone()  
**if** row **is not None**:  
 print(**"Restarting existing crawl. Remove spider.sqlite to start a fresh crawl."**)  
**else** :  
 starturl = input(**'Enter web url or enter: '**)  
 **if** ( len(starturl) < 1 ) : starturl = **'http://www.dr-chuck.com/'  
 if** ( starturl.endswith(**'/'**) ) : starturl = starturl[:-1]  
 web = starturl  
 **if** ( starturl.endswith(**'.htm'**) **or** starturl.endswith(**'.html'**) ) :  
 pos = starturl.rfind(**'/'**)  
 web = starturl[:pos]  
  
 **if** ( len(web) > 1 ) :  
 cur.execute(**'INSERT OR IGNORE INTO Webs (url) VALUES ( ? )'**, ( web, ) )  
 cur.execute(**'INSERT OR IGNORE INTO Pages (url, html, new\_rank) VALUES ( ?, NULL, 1.0 )'**, ( starturl, ) )  
 conn.commit()  
  
*# Get the current webs*cur.execute(**'''SELECT url FROM Webs'''**)  
webs = list()  
**for** row **in** cur:  
 webs.append(str(row[0]))  
  
print(webs)  
  
many = 0  
**while True**:  
 **if** ( many < 1 ) :  
 sval = input(**'How many pages:'**)  
 **if** ( len(sval) < 1 ) : **break** many = int(sval)  
 many = many - 1  
  
 cur.execute(**'SELECT id,url FROM Pages WHERE html is NULL and error is NULL ORDER BY RANDOM() LIMIT 1'**)  
 **try**:  
 row = cur.fetchone()  
 *# print row* fromid = row[0]  
 url = row[1]  
 **except**:  
 print(**'No unretrieved HTML pages found'**)  
 many = 0  
 **break** print(fromid, url, end=**' '**)  
  
 *# If we are retrieving this page, there should be no links from it* cur.execute(**'DELETE from Links WHERE from\_id=?'**, (fromid, ) )  
 **try**:  
 document = urlopen(url, context=ctx)  
  
 html = document.read()  
 **if** document.getcode() != 200 :  
 print(**"Error on page: "**,document.getcode())  
 cur.execute(**'UPDATE Pages SET error=? WHERE url=?'**, (document.getcode(), url) )  
  
 **if 'text/html'** != document.info().get\_content\_type() :  
 print(**"Ignore non text/html page"**)  
 cur.execute(**'DELETE FROM Pages WHERE url=?'**, ( url, ) )  
 conn.commit()  
 **continue** print(**'('**+str(len(html))+**')'**, end=**' '**)  
  
 soup = BeautifulSoup(html, **"html.parser"**)  
 **except** KeyboardInterrupt:  
 print(**''**)  
 print(**'Program interrupted by user...'**)  
 **break  
 except**:  
 print(**"Unable to retrieve or parse page"**)  
 cur.execute(**'UPDATE Pages SET error=-1 WHERE url=?'**, (url, ) )  
 conn.commit()  
 **continue** cur.execute(**'INSERT OR IGNORE INTO Pages (url, html, new\_rank) VALUES ( ?, NULL, 1.0 )'**, ( url, ) )  
 cur.execute(**'UPDATE Pages SET html=? WHERE url=?'**, (memoryview(html), url ) )  
 conn.commit()  
  
 *# Retrieve all of the anchor tags* tags = soup(**'a'**)  
 count = 0  
 **for** tag **in** tags:  
 href = tag.get(**'href'**, **None**)  
 **if** ( href **is None** ) : **continue** *# Resolve relative references like href="/contact"* up = urlparse(href)  
 **if** ( len(up.scheme) < 1 ) :  
 href = urljoin(url, href)  
 ipos = href.find(**'#'**)  
 **if** ( ipos > 1 ) : href = href[:ipos]  
 **if** ( href.endswith(**'.png'**) **or** href.endswith(**'.jpg'**) **or** href.endswith(**'.gif'**) ) : **continue  
 if** ( href.endswith(**'/'**) ) : href = href[:-1]  
 *# print href* **if** ( len(href) < 1 ) : **continue** *# Check if the URL is in any of the webs* found = **False  
 for** web **in** webs:  
 **if** ( href.startswith(web) ) :  
 found = **True  
 break  
 if not** found : **continue** cur.execute(**'INSERT OR IGNORE INTO Pages (url, html, new\_rank) VALUES ( ?, NULL, 1.0 )'**, ( href, ) )  
 count = count + 1  
 conn.commit()  
  
 cur.execute(**'SELECT id FROM Pages WHERE url=? LIMIT 1'**, ( href, ))  
 **try**:  
 row = cur.fetchone()  
 toid = row[0]  
 **except**:  
 print(**'Could not retrieve id'**)  
 **continue** *# print fromid, toid* cur.execute(**'INSERT OR IGNORE INTO Links (from\_id, to\_id) VALUES ( ?, ? )'**, ( fromid, toid ) )  
  
  
 print(count)  
  
cur.close()

sprank.py

**import** sqlite3  
  
conn = sqlite3.connect(**'spider.sqlite'**)  
cur = conn.cursor()  
  
*# Find the ids that send out page rank - we only are interested  
# in pages in the SCC that have in and out links*cur.execute(**'''SELECT DISTINCT from\_id FROM Links'''**)  
from\_ids = list()  
**for** row **in** cur:   
 from\_ids.append(row[0])  
  
*# Find the ids that receive page rank*to\_ids = list()  
links = list()  
cur.execute(**'''SELECT DISTINCT from\_id, to\_id FROM Links'''**)  
**for** row **in** cur:  
 from\_id = row[0]  
 to\_id = row[1]  
 **if** from\_id == to\_id : **continue  
 if** from\_id **not in** from\_ids : **continue  
 if** to\_id **not in** from\_ids : **continue** links.append(row)  
 **if** to\_id **not in** to\_ids : to\_ids.append(to\_id)  
  
*# Get latest page ranks for strongly connected component*prev\_ranks = dict()  
**for** node **in** from\_ids:  
 cur.execute(**'''SELECT new\_rank FROM Pages WHERE id = ?'''**, (node, ))  
 row = cur.fetchone()  
 prev\_ranks[node] = row[0]  
  
sval = input(**'How many iterations:'**)  
many = 1  
**if** ( len(sval) > 0 ) : many = int(sval)  
  
*# Sanity check***if** len(prev\_ranks) < 1 :   
 print(**"Nothing to page rank. Check data."**)  
 quit()  
  
*# Lets do Page Rank in memory so it is really fast***for** i **in** range(many):  
 *# print prev\_ranks.items()[:5]* next\_ranks = dict();  
 total = 0.0  
 **for** (node, old\_rank) **in** list(prev\_ranks.items()):  
 total = total + old\_rank  
 next\_ranks[node] = 0.0  
 *# print total  
  
 # Find the number of outbound links and sent the page rank down each* **for** (node, old\_rank) **in** list(prev\_ranks.items()):  
 *# print node, old\_rank* give\_ids = list()  
 **for** (from\_id, to\_id) **in** links:  
 **if** from\_id != node : **continue** *# print ' ',from\_id,to\_id* **if** to\_id **not in** to\_ids: **continue** give\_ids.append(to\_id)  
 **if** ( len(give\_ids) < 1 ) : **continue** amount = old\_rank / len(give\_ids)  
 *# print node, old\_rank,amount, give\_ids* **for** id **in** give\_ids:  
 next\_ranks[id] = next\_ranks[id] + amount  
   
 newtot = 0  
 **for** (node, next\_rank) **in** list(next\_ranks.items()):  
 newtot = newtot + next\_rank  
 evap = (total - newtot) / len(next\_ranks)  
  
 *# print newtot, evap* **for** node **in** next\_ranks:  
 next\_ranks[node] = next\_ranks[node] + evap  
  
 newtot = 0  
 **for** (node, next\_rank) **in** list(next\_ranks.items()):  
 newtot = newtot + next\_rank  
  
 *# Compute the per-page average change from old rank to new rank  
 # As indication of convergence of the algorithm* totdiff = 0  
 **for** (node, old\_rank) **in** list(prev\_ranks.items()):  
 new\_rank = next\_ranks[node]  
 diff = abs(old\_rank-new\_rank)  
 totdiff = totdiff + diff  
  
 avediff = totdiff / len(prev\_ranks)  
 print(i+1, avediff)  
  
 *# rotate* prev\_ranks = next\_ranks  
  
*# Put the final ranks back into the database*print(list(next\_ranks.items())[:5])  
cur.execute(**'''UPDATE Pages SET old\_rank=new\_rank'''**)  
**for** (id, new\_rank) **in** list(next\_ranks.items()) :  
 cur.execute(**'''UPDATE Pages SET new\_rank=? WHERE id=?'''**, (new\_rank, id))  
conn.commit()  
cur.close()

spdump.py

**import** sqlite3  
  
conn = sqlite3.connect(**'spider.sqlite'**)  
cur = conn.cursor()  
  
cur.execute(**'''SELECT COUNT(from\_id) AS inbound, old\_rank, new\_rank, id, url   
 FROM Pages JOIN Links ON Pages.id = Links.to\_id  
 WHERE html IS NOT NULL  
 GROUP BY id ORDER BY inbound DESC'''**)  
  
count = 0  
**for** row **in** cur :  
 **if** count < 50 : print(row)  
 count = count + 1  
print(count, **'rows.'**)  
cur.close()

spjson.py

**import** sqlite3  
  
conn = sqlite3.connect(**'spider.sqlite'**)  
cur = conn.cursor()  
  
print(**"Creating JSON output on spider.js..."**)  
howmany = int(input(**"How many nodes? "**))  
  
cur.execute(**'''SELECT COUNT(from\_id) AS inbound, old\_rank, new\_rank, id, url   
 FROM Pages JOIN Links ON Pages.id = Links.to\_id  
 WHERE html IS NOT NULL AND ERROR IS NULL  
 GROUP BY id ORDER BY id,inbound'''**)  
  
fhand = open(**'spider.js'**,**'w'**)  
nodes = list()  
maxrank = **None**minrank = **None  
for** row **in** cur :  
 nodes.append(row)  
 rank = row[2]  
 **if** maxrank **is None or** maxrank < rank: maxrank = rank  
 **if** minrank **is None or** minrank > rank : minrank = rank  
 **if** len(nodes) > howmany : **break  
  
if** maxrank == minrank **or** maxrank **is None or** minrank **is None**:  
 print(**"Error - please run sprank.py to compute page rank"**)  
 quit()  
  
fhand.write(**'spiderJson = {"nodes":[\n'**)  
count = 0  
map = dict()  
ranks = dict()  
**for** row **in** nodes :  
 **if** count > 0 : fhand.write(**',\n'**)  
 *# print row* rank = row[2]  
 rank = 19 \* ( (rank - minrank) / (maxrank - minrank) )   
 fhand.write(**'{'**+**'"weight":'**+str(row[0])+**',"rank":'**+str(rank)+**','**)  
 fhand.write(**' "id":'**+str(row[3])+**', "url":"'**+row[4]+**'"}'**)  
 map[row[3]] = count  
 ranks[row[3]] = rank  
 count = count + 1  
fhand.write(**'],\n'**)  
  
cur.execute(**'''SELECT DISTINCT from\_id, to\_id FROM Links'''**)  
fhand.write(**'"links":[\n'**)  
  
count = 0  
**for** row **in** cur :  
 *# print row* **if** row[0] **not in** map **or** row[1] **not in** map : **continue  
 if** count > 0 : fhand.write(**',\n'**)  
 rank = ranks[row[0]]  
 srank = 19 \* ( (rank - minrank) / (maxrank - minrank) )   
 fhand.write(**'{"source":'**+str(map[row[0]])+**',"target":'**+str(map[row[1]])+**',"value":3}'**)  
 count = count + 1  
fhand.write(**']};'**)  
fhand.close()  
cur.close()  
  
print(**"Open force.html in a browser to view the visualization"**)