Program 1, TCSS 480 Autumn 2017
Due Friday, Nov. 3, 2017, noon

OBJECTIVE

The objective of this assignment is to apply your newly learned Python knowledge to build a slightly larger application. The assignment is individual and will require the research of Python built-in libraries.

ASSIGNMENT DESCRIPTION

Given the initial text file with URLs, you are to create a web crawler that searches Web pages for outside / absolute links and processes the required number of pages in a concurrent fashion. You need to generate a csv file containing a map of the URL search space and a word cloud consisting of words used in link names as your output.

Input: a txt file named `urls.txt` (hardcode the file name) with several URLs (one URL per line). Sample provided – note that the last line of the input file is empty.

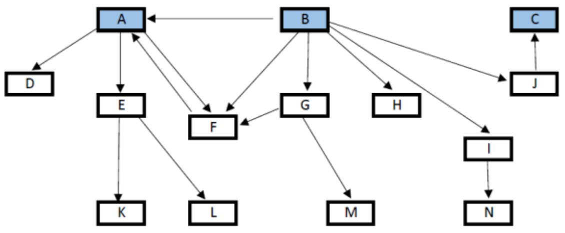
Web crawler: for each of the URLs provided in the initial file, find all the absolute links (as defined below) and the names associated with them, and then go through the pages that are linked from the main URLs until you either process the required number of pages or exhaust the links. An absolute link defines a specific location of the Web file or document including the protocol, the domain name, the directory/s (if any), and the name of the document itself (if any), e.g.
`<a href = "http://www.domain.com/folder1/folder1a/pagename.html">Some text</a>`
In this example, `http://www.domain.com/folder1/folder1a/pagename.html` is the absolute link, whereas `Some text` is the name given to the link. An absolute link may be missing the actual page name, e.g. `http://www.domain.com/folder1/folder1a` or a folder name e.g. `http://www.domain.com/` but MUST contain the domain name. Any link missing the domain name is NOT an absolute link and should NOT be considered, e.g `<a href="somepage.html">link text</a>` is NOT an absolute link.

To simplify processing, assume there are no attributes listed between `a` and `href` tags. However, remember there may or may not be whitespaces around the `=` sign. Assume that the actual website address can only begin with `http` or `https`. You are to use regular expressions to find the links in the HTML document.

In order to process webpages, use `urllib.request` module. The `request` module handles opening and reading of URLs. Once the URL is open, you can read the webpage as if you were reading data from a plain text file using a `for` loop or using `read` and `readline` (returning `bytes` object, not strings), e.g. the code below reads the main page for our Institute of Technology and echo print its HTML to the screen. If needed, you can typecast `bytes` object to a string: in the code below, `pageText` is of type `bytes`, so to store its string version into a variable you may use something like `pt = str(pageText)`

```
import urllib.request
page = urllib.request.urlopen('http://www.tacoma.uw.edu/institute-technology/institute-
technology')
pageText = page.read()
print(pageText)
```

Given the starting nodes in the provided text file, the "crawl" space is to be constructed in the breadth-first order. Example:



If the blue nodes are the starting nodes, then the letters denote the order in which a "crawl" space is to be constructed A-Z (first starting nodes, then all of their immediate children, then the children of children unless there is a cycle, etc.).

Output: Generate a csv file that shows the "crawl" space as:

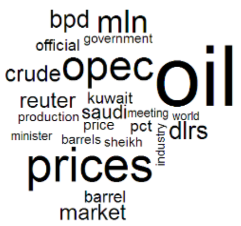
Page	**Links**
A	D, E, F
B	A, F, G, H, I, J
C	
D	
E	K, L
F	A
G	F, M
H	
I	N
J	C
K	
L	
M	
N	

and includes additional information at the bottom that specifies the Web page with the highest number of links pointing to it, in our example F. There may be a possibility that there is a tie for the top and you need to list all pages that tie for the title of being most popular. The upper bound on the crawler ought to be 50 processed pages – this numerical limit should be applied to the pages listed in the column *Page* above and not to the overall number of links. Your csv output file should list actual URLs.

Concurrency: It is up to you to decide how to specifically implement concurrency in this assignment but you should provide both data and task concurrency. It should be obvious that the web crawler processing could be parallelized so that while one process is searching one page and congregating the results, another process could be searching another page. In addition, writing to a file and creating a word cloud should happen in parallel as well. In order to support concurrency, you will need to research and experiment with *multiprocessing* module in depth.

Word Cloud: In addition, you are to generate a word cloud showing 12 most popular words contained in link text (if < 12 words are present in your cloud, then print them all). Cases should be ignored, i.e. *prices* and *Prices* should be considered the same word. You should print the cloud as a dictionary (sorted in decreasing order by the value component) to the console and then in a separate window using graphics. Example:

{'oil' : 9, 'prices': 7, 'opec': 6, 'mln': 4} etc.



SUBMISSION AND GRADING NOTES

You are to submit your finished script through *Canvas* and the test document only. Name your script `pr1.py`. Your code needs to include comments, use consistent coding style, and has to be compatible with Python >= 3.5. You are only allowed to use Python built-in modules (i.e. do NOT use any module that requires additional download).

Assignments that do not meet these criteria will receive a grade of 0. If you decide to provide a test document, upload it as txt or pdf, filename does not matter.

This is an individual assignment – you are NOT allowed to work on it with any other student or share your code with others. This assignment is NOT subject to a peer review process and will be graded by the instructor or a grader.

The code will be graded based on its correctness by running it on instructor/grader designed test cases. Then the code will be looked over for anything that violates good coding practice (e.g. non-meaningful variable names, lack of code modularization, the use of read() method). If you want to ensure you are given credit even if you break some of our test cases, provide thorough comments and a text file that explains your testing strategy. The file should provide a short narrative of how you went about the testing process and give a table of test cases, e.g.

Test case	**Reason for the test**	**Expected Outcome**
give Web address/s	To check what happens when no absolute links exist	The csv file contains only starting Web page addresses
give Web address/s	Testing the word cloud	Cloud dictionary
…	…	…

You will not receive any points for your test plan but it may prove beneficial to you in the grading process.

Note that you may turn in this assignment late, with no penalty, with the deadline of Nov. 10 at noon. No submission past this deadline will be accepted even if you submit at 12:01 p.m. – 7 days of lateness is more than enough.

Grading points will be distributed in the following fashion:

- Sequential Web crawler 30 pts
    - tags recognition
    - graph building logic
    - csv file and highest linked page count
- Word cloud 7.5 pts
    - proper word counts displayed
    - graphics
- Parallelization 7.5 pts
- Coding style and comments 5 pts