1. Implement your crawler according to requirements.

a) Describe the key architecture of your web crawler.

**Answer**:

* I used these libraries: “requests,” “BeautifulSoup”
* The rest of the code is my personal work and not borrowed from anyone else online or in class.
* Procedure in my code: (in order)
  + Create site maps (from robots.txt) to construct ‘disallow’ and ‘allow’ list of urls
  + Send request to the url and get raw content with HTML markup (from ‘allow’ url list)
    - Check for status code of each request. If not ‘200’, push to a list of ‘broken links” and move on.
  + Parse HTML from raw content and make word vector
    - Check for meta tag of noindex. If present, move on; if not present, continue to make word vector

b) Identify and describe your major data structures. Did you use lists, arrays, etc?

**Answer**:

* I use ‘list’ mostly and ‘dictionary’ for some variables.

c) Describe how you implemented “politeness” in your crawler.

**Answer**:

* I hit separate webpage every 3 or 4 seconds plus the time for my crawler to process one webpage after hitting it.

d) Describe in detail how the above is implemented in your code. [25 points]

**Answer**:

my\_crawler = WebCrawl()

* Class WebCrawl
  + functions: (name\_of\_function: *what it does*)
    - create\_site\_map: *get allow/disallow from robots.txt*
    - isNoIndexPage: *check for meta tag noindex*
    - parse\_html\_and\_make\_word\_vector: *parse raw content and make word vector*
    - getTitle: *get titles of all urls*
    - isLocalRobotsTxt: *check for local robots.txt*
    - getRawContentFromTextFile: *get raw content from url*
    - crawl: *run the web crawler*
  + variables: (local variable: type – *description*)
    - N: number - *limit of number of documents needed to parse*
    - domain:list – *domain to crawl*
    - raw\_content: list – *raw content with HTML markup*
    - content\_dict: dict – *dictionary in the format of {url: raw content} for duplicate detection*
    - word\_vector: list/dict – *to store words indexed*
    - disallow: list – *disallow list to crawl*
    - allow: list – *allow list to crawl*
    - broken\_links: list – *broken links, urls that don’t give status code of ‘200’*
    - non\_text\_urls: list – *urls that don’t have extension of text file (specified in requirements)*
    - titles: dict – *dictionary of* *text inside TITLE tag of all pages (in both ‘allow’ and ‘disallow’) in the format of {url: title}*
    - num\_of\_documents\_indexed: number – *number of urls with extension of text file that give status code of ‘200’*
    - word\_indexed: number – *number of words indexed*

2. Use your crawler to list the URL and <TITLE> of all pages in the test data and report all links going out of the test data (i.e. items you must not crawl). [10 points]

CS7337: determine if the page has a html meta tag of noindex. If so, do not index this page, but include it in your listing.

**Answer**:

* my\_crawler.titles gives back the dictionary of titles and their urls.
* my\_crawler.disallow gives back the disallow directories and urls.
* I accidentally implemented checking for noindex page. My crawler doesn’t index that page but it is in my ‘allow’ list of urls.

3. Implement exact duplicate content detection. List the URLs of pages that contain already seen content. Do not index the duplicate pages. [10 points]

**Answer**: I will construct a dictionary of raw\_content of indexed urls, and look up every time I crawl a webpage

if self.raw\_content not in self.content\_dict.values():

self.content\_dict[url] = self.raw\_content

else: #if raw content exists in indexed pages, move on

continue

4. Use your crawler to list all broken links within the test data. [10 points]

**Answer**:

my\_crawler = WebCrawl(domain, N)

my\_crawler.broken\_links # broken links

5. List the URLs of non-text files that are referenced in the test data. [10 points]

Answer:

6. Your crawler must save each word and position from each page of type (.txt, .htm, .html, .php). Make sure that you do not save HTML markup. Unless you provide a different definition, you can assume a word is a string of non-space character(s), beginning with an alphabetic character. If a token does not begin with an alphabetic character, remove those until it is alphabetic or null. You may decide if it can contain special characters, but the last character of a word is either alphabetic or numeric. Perform case insensitive matching. Start with an empty dictionary and add words as they are encountered. In this process, give each page a unique document ID. Further, the output of this step will generate a term-document frequency matrix. Your program may generate the data to be further processed in a spreadsheet (Excel or equivalent).

a) What is your definition of “word”?

**Answer**: My definition is the same as default: *a string of non-space character.*

b) How many documents are indexed?

**Answer**:

c) How many words are indexed?

**Answer**:

d) Generate the term-document frequency matrix. [25 points]

**Answer**:

7. Report the 20 most common words with its document frequency. [10 points]