**1. Project Report:**

**1.1 Description of search engine, including description of main components.**

The search engine provides by default top 5 URL’s based on search query.

If the final.py program is run an Enter query option is displayed, the user enter search string/query in the that field provided. When user click enter button, one can see top 5 related URL’s (based on cosine similarity) below on the console.

**Main Components:**

**Web crawling:**

**Main Functionality:** UNT website is traversed using breadth first strategy and keep track of already visited URL’s to make sure that they are not already visited. All the visited urls are stored in a list, and those url data is read and both urls and the url data are stored in the two different files on the desktop folder.

**Implementation**: The read url data is stored in a file name crawledurls.txt on the desktop. That dataset is opened and tokenization, stemming and stop words removal is done on that. Now that data is compared with the query string and the resultant filenames are given.

To display the urls of top 5 results, the visited urls are stored in urls folder on the desktop. The urls are stored in a list and then compared with the results of the text document and the respective urls are displayed.

Crawler.py has above crawling functionality implementations.

**Indexing:**

**Main Functionality:** The inverted index contains all tokens of vocabulary and the documents in which that token appears and frequency of token in that document.

**Implementation**: On each document or webpage retrieved from crawling, we perform tokenization, stemming, removing stop words and store each token as key and value in dictionary which contain document as key and frequency of token in that document.

Final.py file has implementation of creating an inverted index and performing preprocessing step for the documents.

**Using query retrieving url:**

**Main Functionality:** Retrieving url using user input string and performing preprocessing based on the url document and displaying the url on the console.

**Implementation**: Taking query from user input and split query into tokens using space as delimiter and perform preprocessing step (removing stop words and stem words) and store in a dictionary where tokens are stored as keys and token frequency as values.

Final.py file contains code for query retrieval and processing.

**Ranking the documents:**

**Main Functionality:** Ranking the documents using cosine similarity between documents and query

**Implementation**: Calculated term frequency, inverted document frequency for both query and documents for tokens that are present in query and also calculated document length and query length required for computing cosine similarity of document and query. Cosine similarity of each document for a given query is stored in a list. In that list the values are sorted in decreasing order based on cosine similarity values.

Final.py file has above ranking functionality implementation.

**Output:**

**Main Functionality:** The input string “Enter the query” is for giving the query and after preprocessing ranked documents are displayed in descending order.

**Implementation**: In the input string, query has a textfield where user writes a query and when click on “enter” button, and the top results are displayed and we retrieve top 5 documents and their related URL’s from documents based on cosine similarity value, we retrieve all URL’s related to query and displayed in text area.

Final.py and Cosinesim.java files have above implementation.

**1.2 Discuss Main Challenges encountered in building search engine.**

**Avoiding duplicate urls:**

**Problem:** Perform web traversal of website using breadth first strategy and need to avoid traversal of already visited URL’s.

**Solution:** The initially crawled urls are appended to a list and while crawling the second phase the urls that already exists in the list are not appended and the unique urls are appended.

**Storing URL of the webpage:**

**Problem:** Writing only the downloaded url into a file of visited urls.

**Solution**: A file is created and the urls that are downloaded are only written into the urls, the web page that does not allow to download it’s content is removed from the visited list.

**Displaying on console:**

**Problem:** Since we check the cosine similarity with the document printing the url on console is a difficult task.

**Solution:** Using the split and range functions for getting just the file name, since the file name is stored as a number and the urls also follow the same sequence as of the downloaded webpage, the urls are taken into a list and the int value of file is called when the query string is given.

**Displaying output on console:**

**Problem:** Display related URL’s for the query in the console.

**Solution:** Once the ranked webpages are crawled, using them retrieve the urls using int of the file name.

**1.3 Weighting scheme and similarity measure used.**

**Term frequency (tf):** Number of times a particular token repeats in a document.

**Inverted Document Frequency (IDF)**: Log (N/n)

N: Total number of documents

n: Document frequency of a token

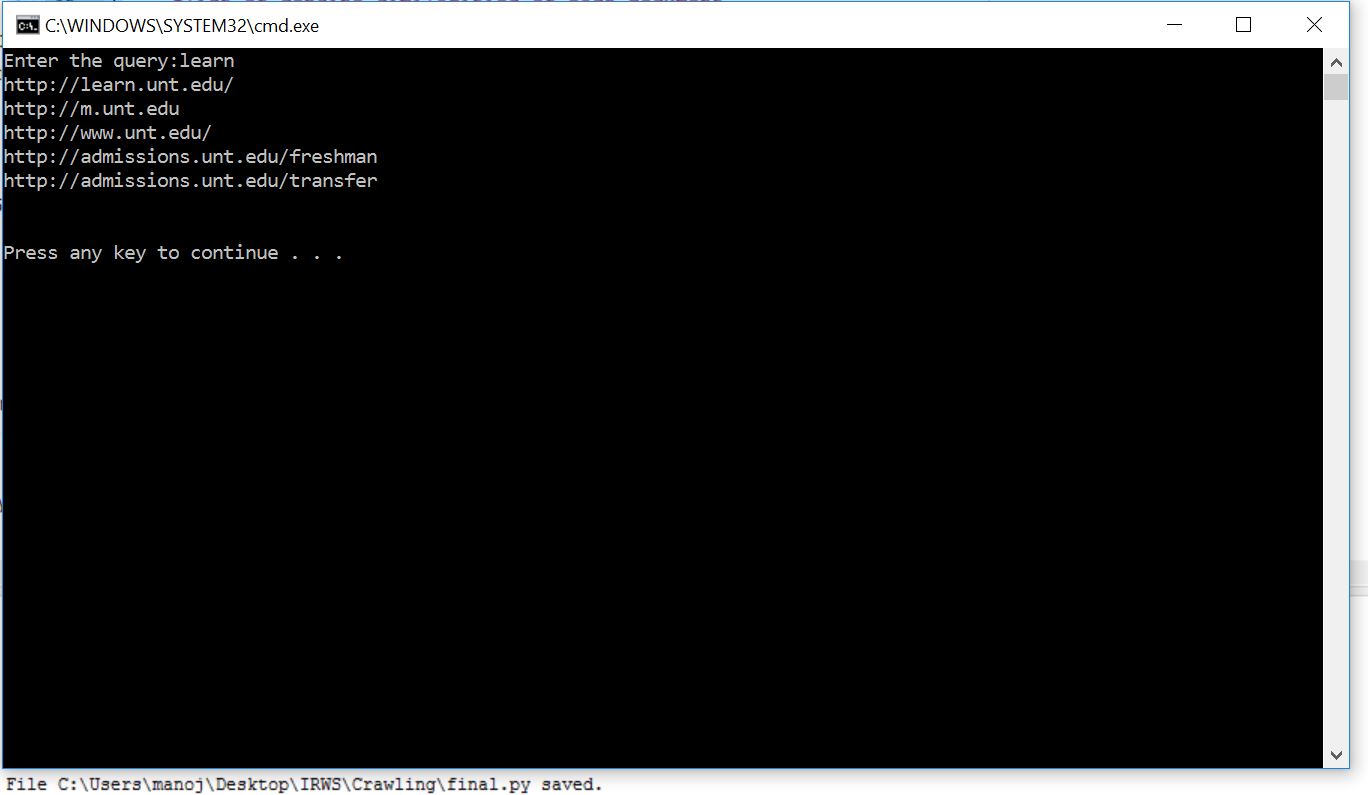
**Cosine Similarity:** Dot product of (D, Q)/ (document length)\*(query length)

For each token of a query calculated tf and idf for both query and document token and performed dot product. Also calculated document length and query length.

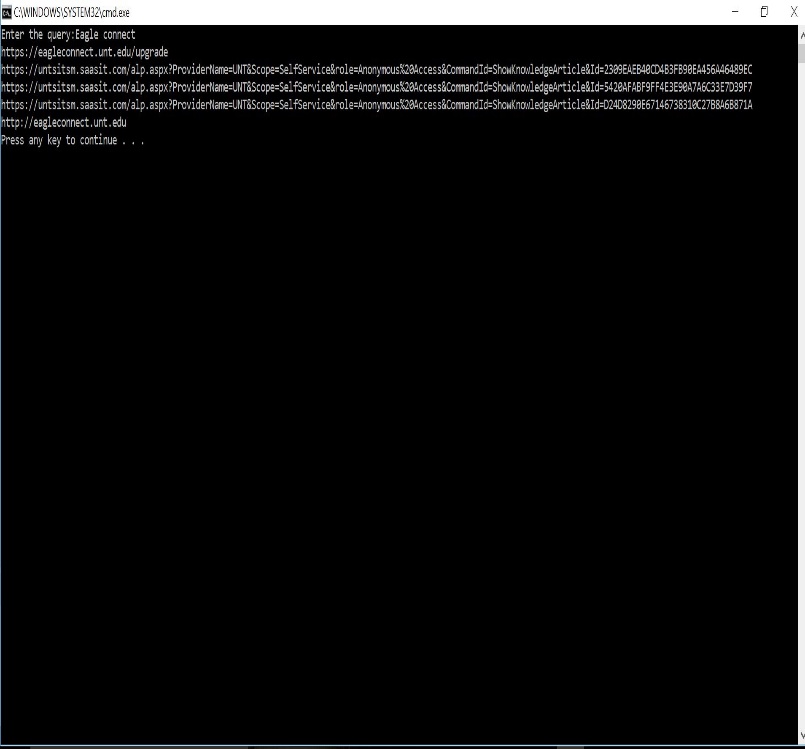
Cosine Similarity for documents (documents which has tokens of query) are sorted and stored in hash map.

Based on cosine similarity values ranking of documents is performed.

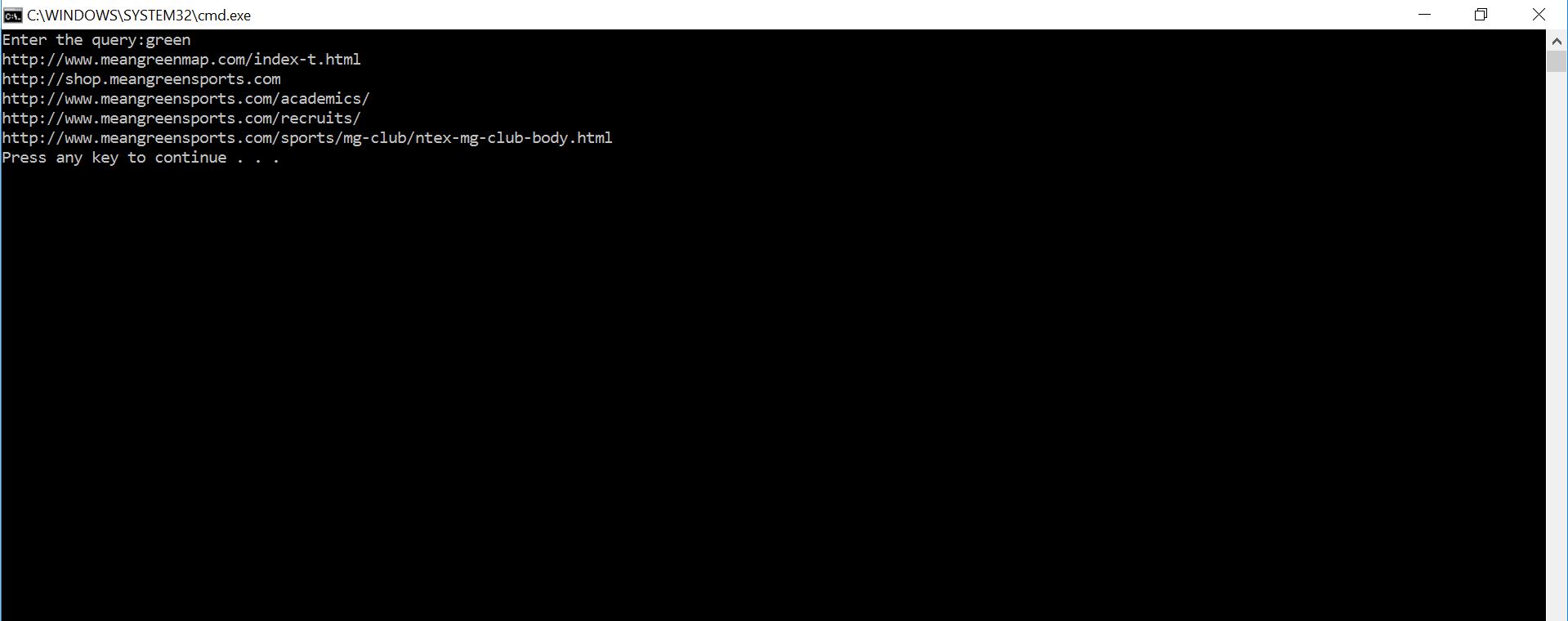
**1.4 Manual evaluation of the top 5 results for three queries of your choice.**

****

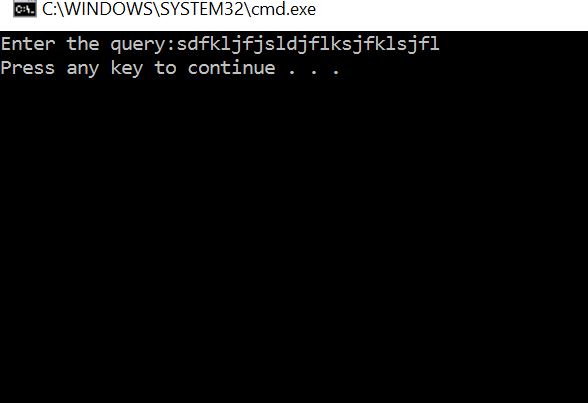
**Screenshot 1: Top 5 URL results for learn query input**



**Screenshot 2: Top 5 URL results for eagle connect query input**

****

**Screenshot 3: Top 5 URL results for green query input**

****

**Screenshot 4: Result for input with no match.**

**2. Software and instructions for how to run the code.**

Python programming language is used.

**Software:**

***Geany IDE:*** For implementing and executing python code

**Instructions to run code:**

**Step 1**: Run the crawler.py code for breadth first crawling. For crawling make sure that there are no files similar to the file to be created. All the crawled and downloaded webpages are stored as number.txt files. Move those files into a folder name Crawledurls. A urls.txt file is downloaded where it contains the urls of the webpages downloaded correspondingly.

**Step 2:** Update those file names if changed in the final.py. Give the path of the data set, stopwords and the urls as in save din the system. And the the final.py file.

Stop words:

stopset = open("C:\\Users\\manoj\\Desktop\\IRWS\\Crawling\\stopwords.txt").read().split()

URL’s:

urls = open("C:\\Users\\manoj\\Desktop\\IRWS\\Crawling\\urls.txt").read()

Dataset:

datasetPath = "C:\\Users\\manoj\\Desktop\\IRWS\\Crawling\\Crawledurls\\"

**Step 3:** Set the range for the displaying number of urls. For example 5 “for item in results[:5]”

**Step 4:** Now run the final.py file. Once it is run we find a display string ”Enter the query:”. In that display we need enter a string as input from the keyboard.

**Step 5:** Once the query string is entered, the cosine similarity between the query and the documents are found out and the top 5 urls are displayed in the descending order.

**3. Link to Crawled data:**

All crawled data files are zipped and uploaded in blackboard. Folder name for crawled data is urls.

**Reference links:**

<http://stackoverflow.com/questions/11596623/i-dont-understand-encode-and-decode-in-python-2-7-3>

<http://stackoverflow.com/questions/899103/writing-a-list-to-a-file-with-python>

<https://docs.python.org/2/library/codecs.html>

<https://docs.python.org/2/>

<https://gist.github.com/duggalrahul/6548584>