Assignment 2   **Hussam Hallak**

CS834, Information Retrieval, Fall 2017   CS Master's Student

Old Dominion University, Computer Science Dept   Prof: Dr. Nelson

Question 1:

Exercise 4.3:

Try to estimate the number of web pages indexed by two different search engines using the technique described in this chapter. Compare the size estimates from a range of queries and discuss the consistency (or lack of it) of these estimates.

Answer:

I have chosen Google and Bing search engines. The technique described in this chapter uses the formula:

$$N = \frac{f_a \times f_b}{f_{ab}}$$

Where:

a and b are the independent terms

N is the estimated total number of indexed pages

f_a is the number of pages containing the term a

f_b is the number of pages containing the term b

f_{ab} is the number of pages containing both a and b

The two terms in the query need to be as independent as possible, so I have chosen the following queries:

1. Diesel Memory
2. Hairy Phone
3. Computer Manifold
4. Electronic Lumber

1. Diesel Memory: I ran the first query "Diesel Memory" and found the following:

A. Google:

Google found about 207,000,000 results for the term "Diesel".

Figure 1: Query: Diesel, Search Engine: Google

Google found about 377,000,000 results for the term "Memory".

Figure 2: Query: Memory, Search Engine: Google
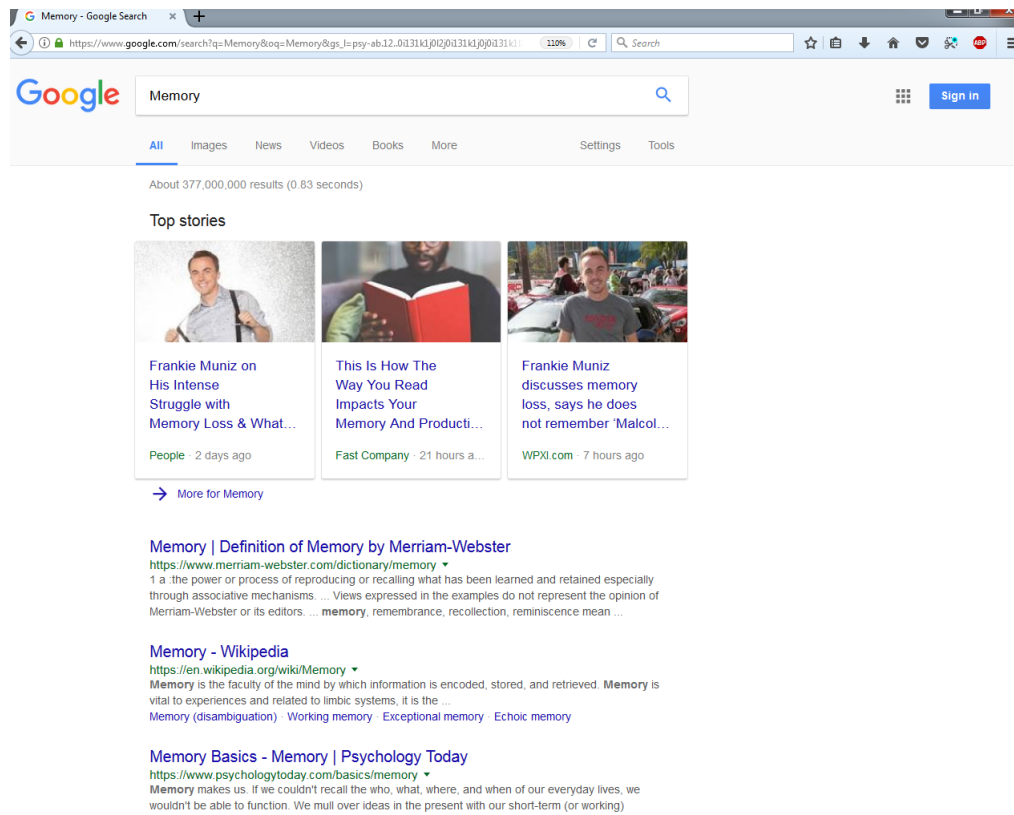
Google found about 36,900,000 results for the query "Diesel Memory".

Figure 3: Query: Diesel Memory, Search Engine: Google

Plugging the results in the formula gives the following:

$$N_{google} = \frac{207,000,000 \times 377,000,000}{36,900,000} = 2114878048.780488 \approx 2114878049$$

B. Bing:

Bing found about 183,000,000 results for the term "Diesel".

Figure 4: Query: Diesel, Search Engine: Bing

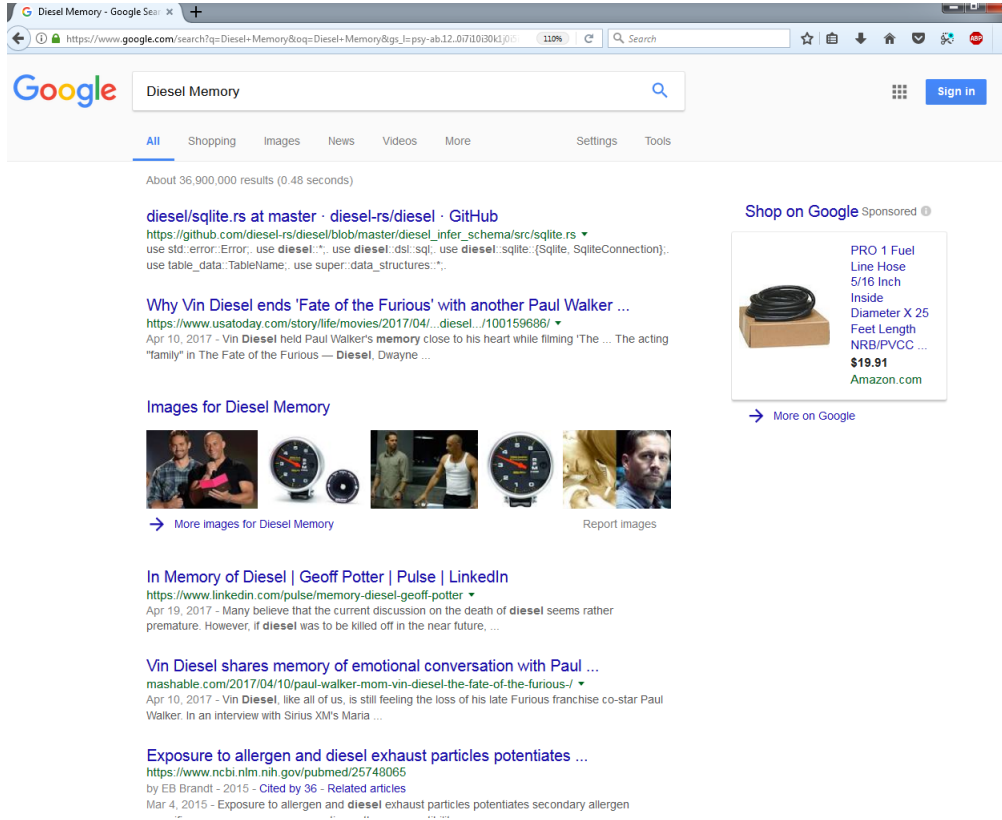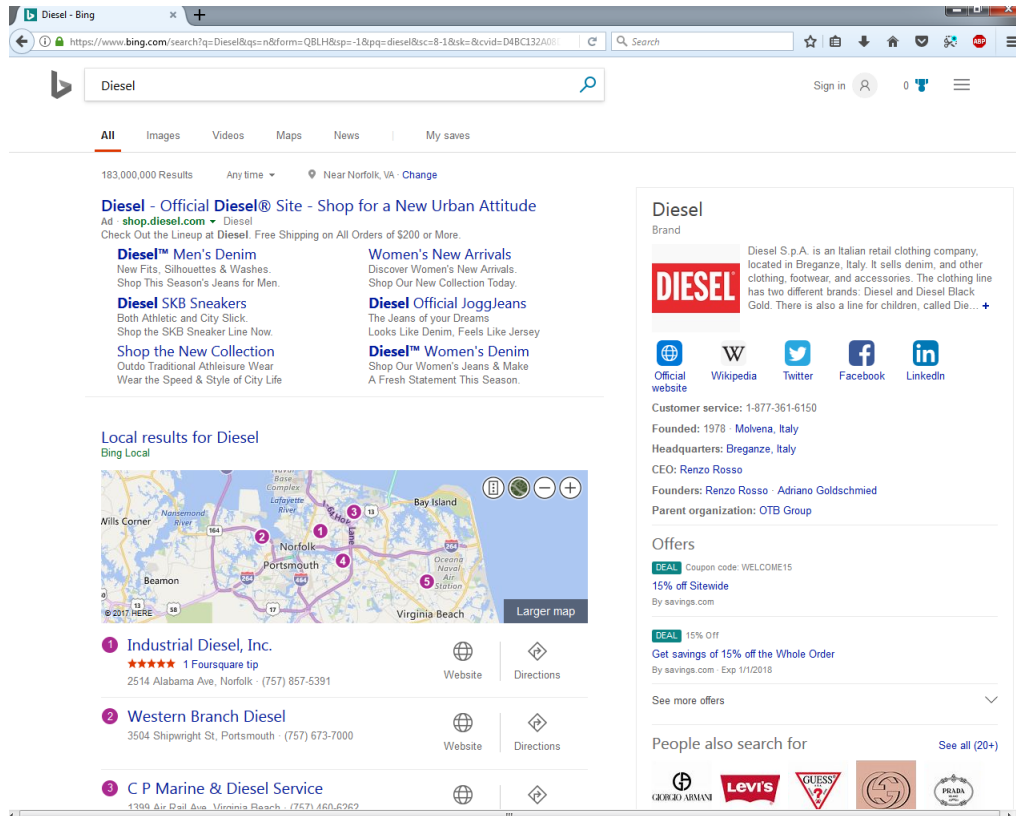Bing found about 167,000,000 results for the term "Memory".

Figure 5: Query: Memory, Search Engine: Bing
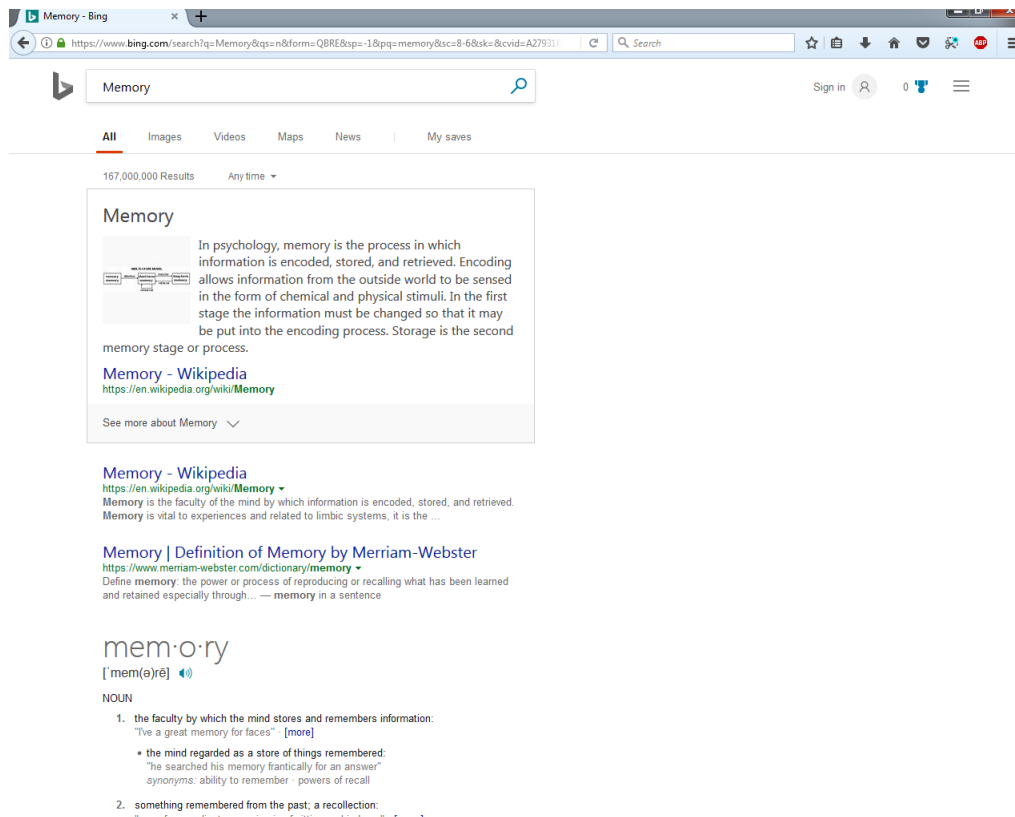
Bing found about 3,220,000 results for the query "Diesel Memory".

Figure 6: Query: Diesel Memory, Search Engine: Bing
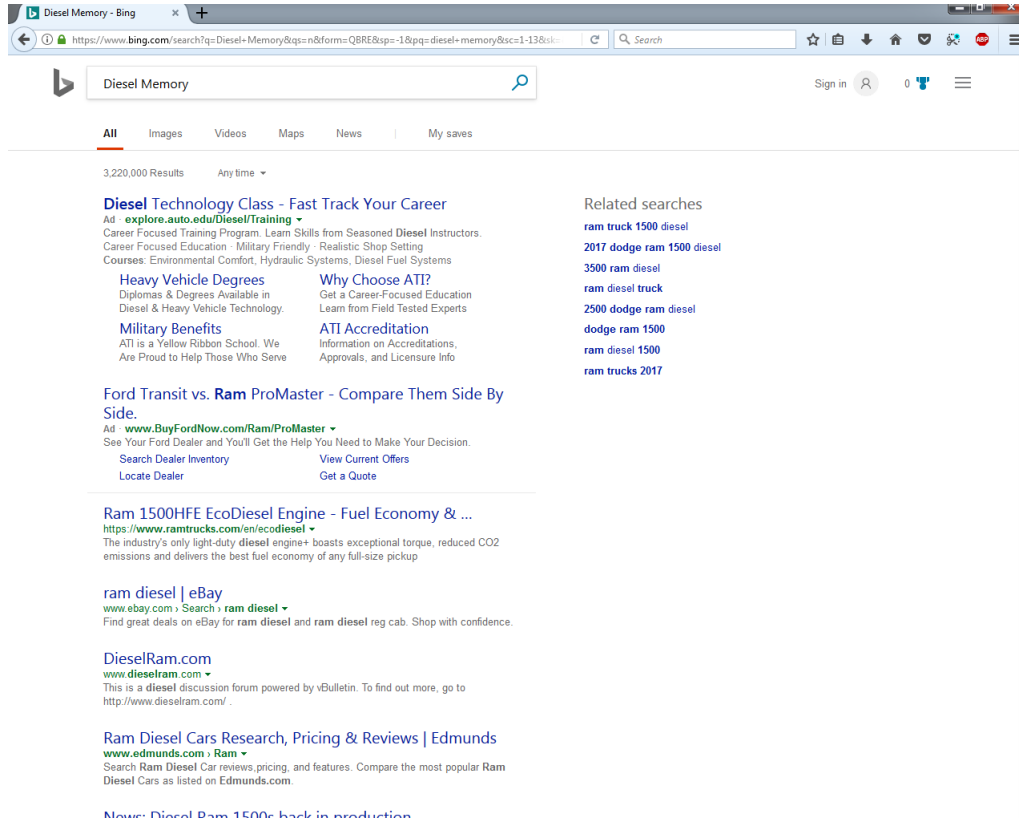
Plugging the results in the formula gives the following:

$$N_{Bing} = \frac{183,000,000 \times 167,000,000}{3,220,000} = 9490993788.819876 \approx 9490993789$$

2. **Hairy Phone:** I ran the second query "Hairy Phone" and found the following:
A. Google:
Google found about 106,000,000 results for the term "Hairy".
Google found about 1,410,000,000 results for the term "Phone".
Google found about 8,780,000 results for the query "Hairy Phone".
Plugging the results in the formula gives the following:

$$N_{google} = \frac{106,000,000 \times 1,410,000,000}{8,780,000} = 17022779043.28018 \approx 17022779043$$

B. Bing:
Bing found about 292,000,000 results for the term "Hairy".
Bing found about 748,000,000 results for the term "Phone".
Bing found about 51,500,000 results for the query "Hairy Phone".
Plugging the results in the formula gives the following:

$$N_{Bing} = \frac{292,000,000 \times 748,000,000}{51,500,000} = 4241087378.640777 \approx 4241087379$$

7

3. Computer Manifold: I ran the third query "Computer Manifold" and found the following:

A. Google:

Google found about 861,000,000 results for the term "Computer".

Google found about 21,500,000 results for the term "Manifold".

Google found about 6,470,000 results for the query "Computer Manifold".

Plugging the results in the formula gives the following:

$$N_{google} = \frac{861,000,000 \times 21,500,000}{6,470,000} = 2861128284.38949 \approx 2861128284$$

B. Bing:

Bing found about 448,000,000 results for the term "Computer".

Bing found about 38,400,000 results for the term "Manifold".

Bing found about 35,100,000 results for the query "Computer Manifold".

Plugging the results in the formula gives the following:

$$N_{Bing} = \frac{448,000,000 \times 38,400,000}{35,100,000} = 490119658.1196581 \approx 490119658$$

4. Electronic Lumber: I ran the forth query "Electronic Lumber" and found the following:

A. Google:

Google found about 1,010,000,000 results for the term "Electronic".

Google found about 43,300,000 results for the term "Lumber".

Google found about 5,720,000 results for the query "Electronic Lumber".

Plugging the results in the formula gives the following:

$$N_{google} = \frac{1,010,000,000 \times 43,300,000}{5,720,000} = 7645629370.629371 \approx 7645629371$$

B. Bing:

Bing found about 289,000,000 results for the term "Electronic".

Bing found about 42,400,000 results for the term "Lumber".

Bing found about 244,000,000 results for the query "Electronic Lumber".

Plugging the results in the formula gives the following:

$$N_{Bing} = \frac{289,000,000 \times 42,400,000}{244,000,000} = 50219672.13114754 \approx 50219672$$

The estimated number of indexed pages in Google as I found are:

2114878049

17022779043

2861128284

7645629371

It is clear that the estimates varied markedly, by few billions, for different queries. I assume that one of the reasons behind that is the difference in "independence level" of the two terms among different queries.

The estimated number of indexed pages in Bing as I found are:

9490993789

4241087379

490119658

50219672

The same inconsistency held for Bing. The estimates varied by few billions.

Question 2:

Exercise 4.6:

Process five Wikipedia documents using the Porter stemmer and the Krovetz stemmer. Compare the number of stems produced and find 10 examples of differences in the stemming that could have an impact on ranking.

Answer:

I have chosen the following documents from WikiSmall collection to process them using Porter and Krovetz stemmers.

Aakrosh_(1998_film).html
Aargau_frank.html
Aaron_Chalmers_3ab5.html
Baba_Budan_67b5.html
Babs_Fafunwa_5f48.html

I wrote a simple python script PKstem.py to process the documents, stem the unique words in them, using Porter and Krovetz stemmers. The stemmers were not implemented from scratch because there are python libraries for both stemmers already.

Listing 1: The content of PKstem.py

```python
import sys
import os
import io
import html2text
from nltk import PorterStemmer
import krovetzstemmer
if __name__ == "__main__":
    if len(sys.argv) < 2:
    print "Usage: python PKstem.py <file1> <file2> <file3> ..."
    print "e.g: python PKstem.py index.html myPage.html default.html some_page.
        html"
    exit()
file_names = set()
for i in range(1,len(sys.argv)):
    file_names.add(sys.argv[i])
porter = PorterStemmer()
krovetz = krovetzstemmer.Stemmer()
output = {}
for file_name in file_names:
    print "Stemming: " + file_name
    h = html2text.HTML2Text()
    h.ignore_links = True
    text = h.handle(u' '.join([line.strip() for line in io.open(file_name, "r",
        encoding="utf-8").readlines()]))
    words = set()
    for word in text.split():
        if word.isalpha():
```

```python
            words.add(word.lower())
    words = sorted(words)
    porter_output = set()
    krovetz_output = set()
    for word in words:
        porter_output.add(porter.stem(word))
        krovetz_output.add(krovetz.stem(word))
    porter_output = sorted(porter_output)
    krovetz_output = sorted(krovetz_output )
    output[file_name] = {}
    output[file_name]['original_text'] = u' '.join(words)
    output[file_name]['porter'] = u' '.join(porter_output)
    output[file_name]['krovetz'] = u' '.join(krovetz_output)
print "Unique words in the original text:"
print "---------------------------------"
print output[file_name]['original_text']
print "Unique stems from Porter stemmer:"
print "---------------------------------"
print output[file_name]['porter']
print "Unique stems from Krovetz stemmer:"
print "---------------------------------"
print output[file_name]['krovetz']
print "The number of unique words in the original text:"
print "------------------------------------------------"
print len(set(output[file_name]['original_text'].split()))
print "The number of unique stems from Porter stemmer:"
print "------------------------------------------------"
print len(set(output[file_name]['porter'].split()))
print "The number of unique stems from Krovetz stemmer:"
print "------------------------------------------------"
print len(set(output[file_name]['krovetz'].split()))
```

The program accepts the document(s) file name(s) as command line argument(s). It is possible to run the program and process all files at once, however, I ran each file separately.

After running the documents, I found the following results:

Aakrosh_(1998_film).html

Unique words: 229

Porter stems: 224

Krovetz stems: 227

Aargau_frank.html

Unique words: 199

Porter stems: 193

Krovetz stems: 193

Aaron_Chalmers_3ab5.html

Unique words: 154

Porter stems: 149

Krovetz stems: 150

Baba_Budan_67b5.html

Unique words: 92

Porter stems: 90

Krovetz stems: 91

Babs_Fafunwa_5f48.html

Unique words: 239

10

Porter stems: 231
Krovetz stems: 235
This is a screen shot of running the program on the first file:

Figure 7: Running PKstem.py

I found the following examples of differences in the stemming that could have an impact on ranking:

Original Text	Porter Stem	Krovetz Stem
daily	daili	daily
changes	chang	change
academic	academ	academic
collaboration	collabor	collaboration
manchester	manchest	manchester
denominations	denomin	denomination
circulating	circul	circulate
article	articl	article
versus	versu	versus
association	associ	association

Question 3:

Exercise 4.8:

Find the 10 Wikipedia documents with the most inlinks. Show the collection of anchor text for those pages.

Answer:

In order to find the 10 Wikipedia documents with the most inlinks, I created a small python script "inlinks.py" that extracts all links found in Wikipedia documents from WikiSmall collection. I stored the links as a key-value pair, where file name is the key and the link is the value. After grouping the links by their target, I counted them and sorted the files by the number of inlinks to find first 10 Wikipedia documents with the most inlinks.

Listing 2: The content of inlinks.py

```python
import os
import sys
import io
import html2text
from bs4 import BeautifulSoup
from tabulate import tabulate

html_files = []
for root, dirs, files in os.walk(os.path.abspath('./en')):
    for file in files:
        if file.endswith('.html'):
            filepath = os.path.join(root, file)
            html_files.append(filepath)

print "Processing files..."
all_links = {}
all_anchor_text = {}
for file in html_files:
    try:
        soup = BeautifulSoup(open(file), 'html.parser')
```

```python
                anchors = soup.find_all('a', href=True)
                for a in anchors:
                    link = a['href']
                    text = a.string or ''
                    if not link.startswith('http'):
                        try:
                            link = os.path.join(os.path.dirname(file), link)
                            link = os.path.abspath(link)
                        except:
                            pass
                    all_links.setdefault(file, [])
                    all_anchor_text.setdefault(file, [])
                    if file != link and link not in all_links[file] and os.path.isfile(
                        link):
                        all_links[file].append(link)
                        all_anchor_text[file].append(text)
        except:
            pass

print "Finding links' frequencies..."
link_frequency = {}
link_text = {}
for source in all_links:
    destinations = all_links[source]
    texts = all_anchor_text[source]

    for index, destination in enumerate (destinations):
        link_frequency.setdefault(destination, 0)
        link_frequency[destination] += 1
        link_text.setdefault(destination, [])
        link_text[destination].append(texts[index])

link_frequency_table = []
for link in link_frequency:
    link_frequency_table.append([link, link_frequency[link]])
link_frequency_table = sorted(link_frequency_table, key=lambda x:x[1], reverse=
    True)
anchor_link_frequency_table = []
for row in link_frequency_table:
    row += [u', '.join(set(link_text[row[0]]))]
    row[0] = os.path.basename(row[0])
    anchor_link_frequency_table.append(row)
link_frequency_table = anchor_link_frequency_table
link_frequency_table = link_frequency_table[:10]

print "Writing the results to output file..."
output = open("output.txt", "w")
output.write(tabulate(link_frequency_table, headers=["File", "Number of inlinks",
     "Anchor text(s)"]))
output.close()
print "Done!"
```
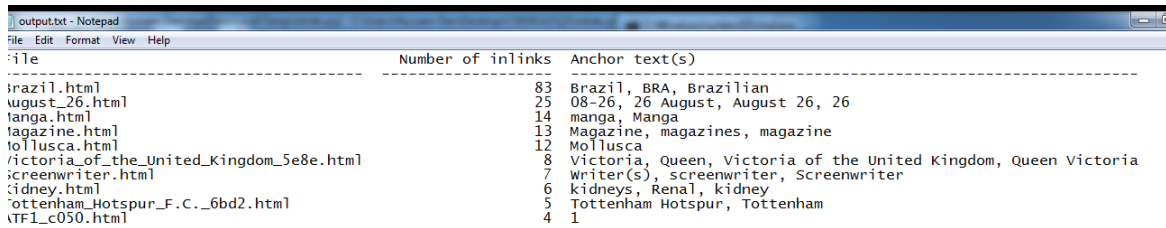
The program writes the result to a file "output.txt". This is a screen shot of the file:

Figure 8: Output of inlinks.py


```
output.txt - Notepad
File  Edit  Format  View  Help
File                                          Number of inlinks  Anchor text(s)
--------------------------------------------  -----------------  --------------------------------------------------------------
Brazil.html                                          83          Brazil, BRA, Brazilian
August_26.html                                       25          08-26, 26 August, August 26, 26
Manga.html                                           14          manga, Manga
Magazine.html                                        13          Magazine, magazines, magazine
Mollusca.html                                        12          Mollusca
Victoria_of_the_United_Kingdom_5e8e.html              8          Victoria, Queen, Victoria of the United Kingdom, Queen Victoria
Screenwriter.html                                     7          Writer(s), screenwriter, Screenwriter
Kidney.html                                           6          kidneys, Renal, kidney
Tottenham_Hotspur_F.C._6bd2.html                      5          Tottenham Hotspur, Tottenham
ATF1_c050.html                                        4          1
```

Question 4:

Exercise 5.8:

Write a program that can build a simple inverted index of a set of text documents. Each inverted list will contain the file names of the documents that contain that word.

Suppose the file A contains the text the quick brown fox, and file B contains the slow blue fox. The output of your program would be:

% ./your-program A B
blue B
brown A
fox A B
quick A
slow B
the A B

Answer:

To create an inverted index, we need to build a table of two columns. The first column contains the words that appear in the set of chosen documents, two in this case. The second column contains the documents in which the words appear.

I created a small python script "inverted.py" that accepts two command line arguments; these are the file names of the documents for which the inverted index will be built. The program stores the unique words in each document as a key-value pair where the document is the key and the words are the value. Swapping the key and the value will generate the inverted index we need. The output is printed on the screen and saved to the file "output.txt".

Listing 3: The content of inverted.py

```python
import io
import os
import sys
import html2text
from tabulate import tabulate

if __name__ == "__main__":
    if len(sys.argv) < 2:
    print "Usage: python inverted.py <file1> <file2> <file3> ..."
    print "e.g: python inverted.py index.html myPage.html default.html some_page.
        html"
    exit()
```

```python
file_names = set()
for i in range(1,len(sys.argv)):
    file_names.add(sys.argv[i])

index = {}
all_words = set()
for file_name in file_names:
    h = html2text.HTML2Text()
    h.ignore_links = True
    text = h.handle(u' '.join([line.strip() for line in io.open(file_name, "r",
        encoding="utf-8").readlines()]))
    words = [word.lower() for word in text.split() if word.isalpha()]
    all_words |= set(words)
    index[file_name.split(os.pathsep)[-1]] = words

inverted_index = {}
for word in all_words:
    files = []
    for file, words in index.items():
        if word in words:
            files.append(file)
    inverted_index[word] = sorted(set(files))

inverted_index_table = []
for word in inverted_index:
    inverted_index_table.append([word, u', '.join(inverted_index[word])])

print tabulate(inverted_index_table, headers=["Word", "Documents"])
output = open("output.txt", "w")
output.write(tabulate(inverted_index_table, headers=["Word", "Documents"]))
output.close()
```
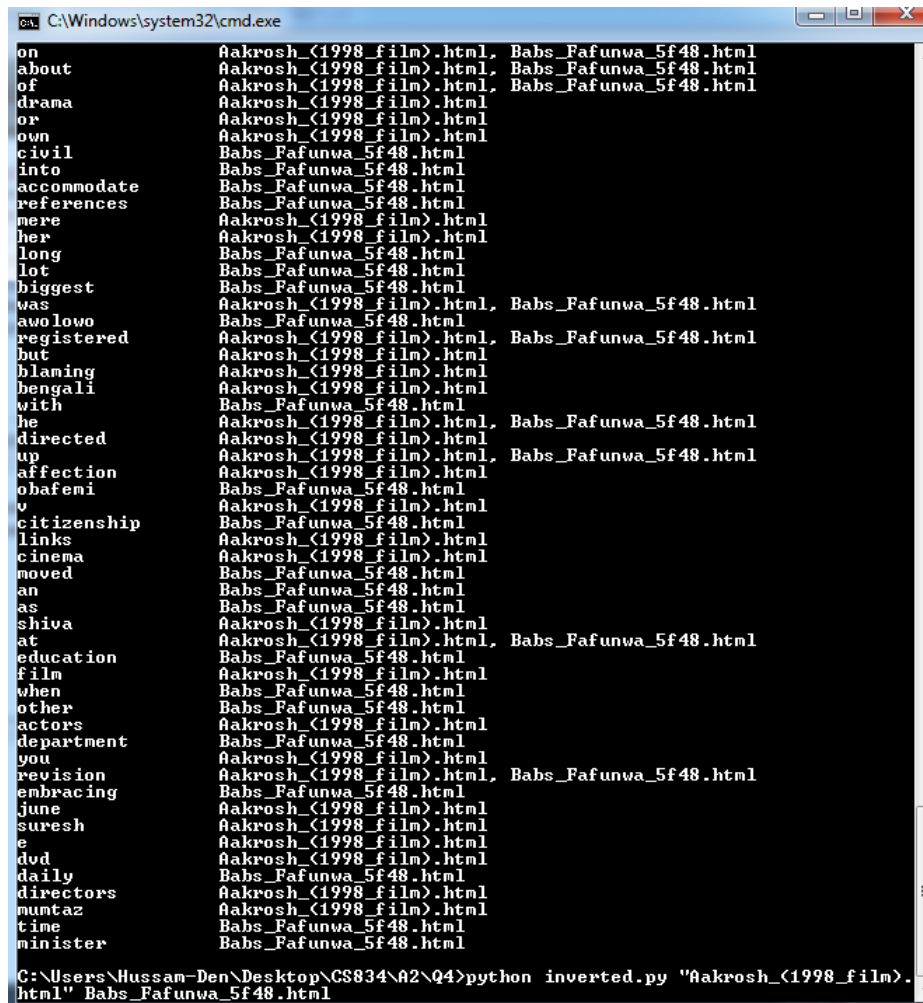
The files I chose to use to build the inverted index are taken from the WikiSmall collection. These files are:

Aakrosh_(1998_film).html

Babs_Fafunwa_5f48.html

This is a screen shot of running the program:

Figure 9: Running inverted.py


```
C:\Windows\system32\cmd.exe

on              Aakrosh_(1998_film).html, Babs_Fafunwa_5f48.html
about           Aakrosh_(1998_film).html, Babs_Fafunwa_5f48.html
of              Aakrosh_(1998_film).html, Babs_Fafunwa_5f48.html
drama           Aakrosh_(1998_film).html
or              Aakrosh_(1998_film).html
own             Aakrosh_(1998_film).html
civil           Babs_Fafunwa_5f48.html
into            Babs_Fafunwa_5f48.html
accommodate     Babs_Fafunwa_5f48.html
references      Babs_Fafunwa_5f48.html
mere            Aakrosh_(1998_film).html
her             Aakrosh_(1998_film).html
long            Babs_Fafunwa_5f48.html
lot             Babs_Fafunwa_5f48.html
biggest         Babs_Fafunwa_5f48.html
was             Aakrosh_(1998_film).html, Babs_Fafunwa_5f48.html
awolowo         Babs_Fafunwa_5f48.html
registered      Aakrosh_(1998_film).html, Babs_Fafunwa_5f48.html
but             Aakrosh_(1998_film).html
blaming         Aakrosh_(1998_film).html
bengali         Aakrosh_(1998_film).html
with            Babs_Fafunwa_5f48.html
he              Aakrosh_(1998_film).html, Babs_Fafunwa_5f48.html
directed        Aakrosh_(1998_film).html
up              Aakrosh_(1998_film).html, Babs_Fafunwa_5f48.html
affection       Aakrosh_(1998_film).html
obafemi         Babs_Fafunwa_5f48.html
v               Aakrosh_(1998_film).html
citizenship     Babs_Fafunwa_5f48.html
links           Aakrosh_(1998_film).html
cinema          Aakrosh_(1998_film).html
moved           Babs_Fafunwa_5f48.html
an              Babs_Fafunwa_5f48.html
as              Babs_Fafunwa_5f48.html
shiva           Aakrosh_(1998_film).html
at              Aakrosh_(1998_film).html, Babs_Fafunwa_5f48.html
education       Babs_Fafunwa_5f48.html
film            Aakrosh_(1998_film).html
when            Babs_Fafunwa_5f48.html
other           Babs_Fafunwa_5f48.html
actors          Aakrosh_(1998_film).html
department      Babs_Fafunwa_5f48.html
you             Aakrosh_(1998_film).html
revision        Aakrosh_(1998_film).html, Babs_Fafunwa_5f48.html
embracing       Babs_Fafunwa_5f48.html
june            Aakrosh_(1998_film).html
suresh          Aakrosh_(1998_film).html
e               Aakrosh_(1998_film).html
dvd             Aakrosh_(1998_film).html
daily           Babs_Fafunwa_5f48.html
directors       Aakrosh_(1998_film).html
mumtaz          Aakrosh_(1998_film).html
time            Babs_Fafunwa_5f48.html
minister        Babs_Fafunwa_5f48.html

C:\Users\Hussam-Den\Desktop\CS834\A2\Q4>python inverted.py "Aakrosh_(1998_film).
html" Babs_Fafunwa_5f48.html
```
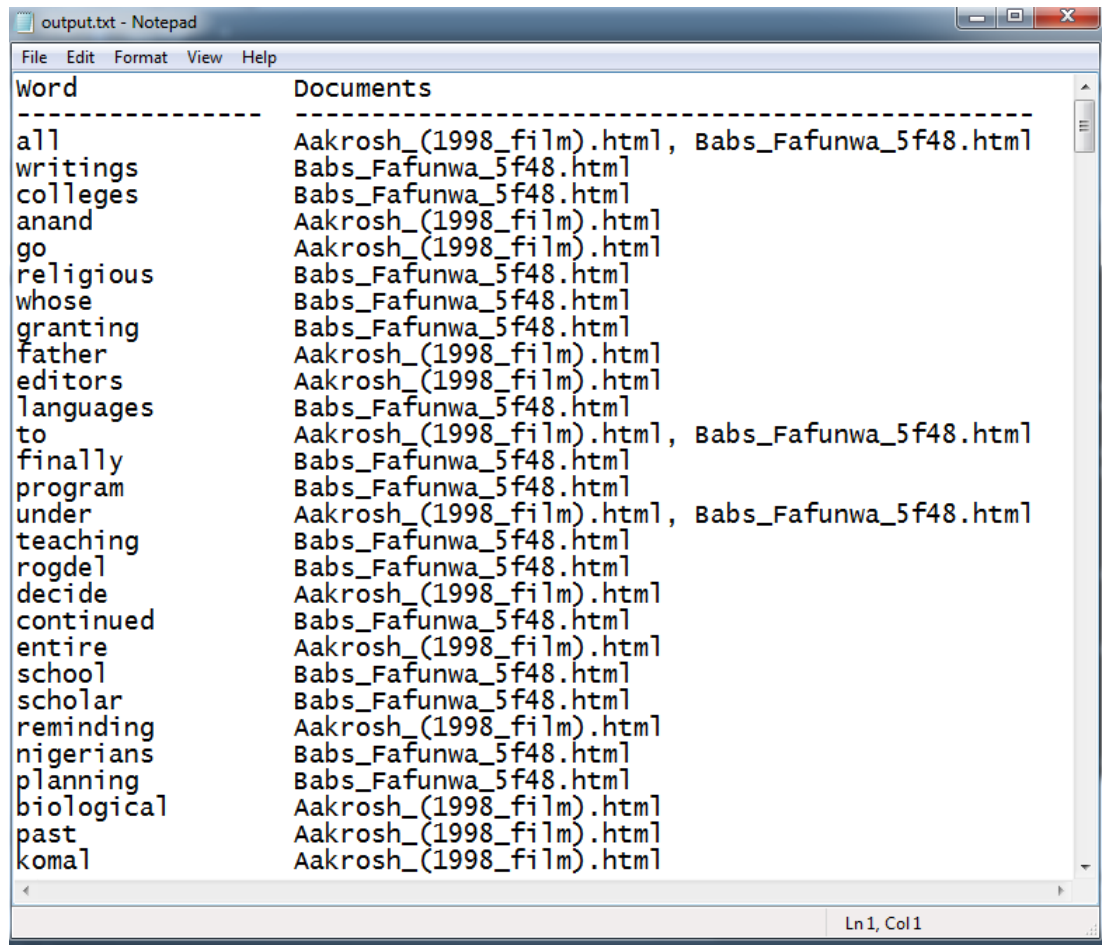
This is a screen shot of the output file "output.txt":

Figure 10: output.txt screen shot

Question 5:

Exercise 5.10:

Suppose a company develops a new unambiguous lossless compression scheme for 2-bit numbers called SuperShrink. Its developers claim that it will reduce the size of any sequence of 2-bit numbers by at least 1 bit. Prove that the developers are lying. More specifically, prove that either:

SuperShrink never uses less space than an uncompressed encoding, or

There is an input to SuperShrink such that the compressed version is larger than the uncompressed input

You can assume that each 2-bit input number is encoded separately.

Answer:

We have 4 possible 2-bit numbers and 3 possible shorter bit sequences. Let's apply the pigeonhole principle, which states that if n items are put into m containers, with n ¿ m, then at least one container must contain more than one item. This means that any

mapping from 2-bit sequences to shorter sequences must have at least 2 sequences compressed to the same shorter sequence. Therefore, when trying to decompress this shorter sequence, mentioned above, it is impossible know which of the original 2-bit sequences it was compressed from.

References

[1] Stackoverflow. https://stackoverflow.com/questions/tagged/python.

[2] Sergey Brin and Larry Page. Google search engine. http://google.stanford.edu.

[3] http://www.dba-oracle.com/oracle_news/news_ebay_massive_oracle.htm

[4] https://www.wired.com/2012/01/amazon-dynamodb/

[5] https://www.youtube.com/watch?v=a0OvgTfF8Pg

[6] https://stackoverflow.com/questions/6526181/write-a-program-that-takes-text-as-input-and-produces-a-program-that-reproduces/6526266#6526266

[7] https://en.wikipedia.org/wiki/Pigeonhole_principle