

Assignment 1

Fall 2016

CS834 Introduction to Information Retrieval

Dr. Michael Nelson

Mathew Chaney

September 20, 2016

Contents

1	Question 1.1	3
1.1	Question	3
1.2	Resources	3
1.3	Methodology	3
1.4	Results	4
2	Question 1.2	5
2.1	Question	5
2.2	Resources	5
2.3	Answer	5
3	Question 3.7	6
3.1	Question	6
3.2	Resources	6
3.3	Answer	6
4	Question 3.8	7
4.1	Question	7
4.2	Resources	7
4.3	Answer	7
5	Question 3.9	8
5.1	Question	8
5.2	Resources	8
5.3	Answer	8
6	Appendix A	9
7	References	12

Listings

1	test directory structure	6
2	site map generator output	6
3	crawler.py output	8
4	crawler.py	9
5	sitemapgen.py	10

List of Tables

1	Relevance Scores	4
---	----------------------------	---

1 Question 1.1

1.1 Question

Think up and write down a small number of queries for a web search engine. Make sure that the queries vary in length (i.e., they are not all one word). Try to specify exactly what information you are looking for in some of the queries. Run these queries on two commercial web search engines and compare the top 10 results for each query by doing relevance judgments. Write a report that answers at least the following questions: What is the precision of the results? What is the overlap between the results for the two search engines? Is one search engine clearly better than the other? If so, by how much? How do short queries perform compared to long queries?

1.2 Resources

The search engines Google [1] and DuckDuckGo [2] were used to obtain the results.

1.3 Methodology

The following search queries were issued to each of the two mentioned search engines:

1. professional skateboarder song
2. daewoo song
3. rollerblade song korea
4. daewon song

The expected relevant pages will contain information regarding the professional skateboarder named Daewon Song. A score is assigned to each search engine for each query using the following equation:

$$\frac{R}{10}$$

Where R the number of relevant documents returned for the given query. Overlap is calculated by using the following equation:

$$\frac{\|G_i \cap D_i\|}{10}$$

Where G_i is the set of pages returned by Google for query i and D_i is the set of pages returned by DuckDuckGo for query i . This result is divided by the size of the overall result set for each query per search engine, ten.

A brief description of the results is provided, with any items that stand out to the reviewer given extra measure. After the detailed listing of results, Table 1 lists the relevancy scores and overlap for each search engine per query.

First Query: “professional skateboarder song”

The relevancy scores from the first query were evenly matched for both search engines, five out of ten relevant documents each. Interestingly, Google provided more social media related results, whereas DuckDuckGo provided a couple celebrity net worth results.

Second Query: “daewoo song”

Trying to throw a curveball at the engine with this one, it is intentionally misspelled. Results for both engines show active error correction and query suggestion functionality, which helps to increase relevance when a user perhaps does not know the correct spelling of an item they wish to find. Google wins this round with eight out of ten versus DuckDuckGo’s result of 6 out of ten.

Third Query: “rollerblade song korea”

Somewhat of another curve ball, although this is *close* to a “correct” query, since rollerblading and skateboarding are similar “extreme” sports. The results, however, show that these types of judgements are not available to a search engine, as both websites returned zero relevant documents pertaining to Daewon Song. Perhaps the error simply lies between the chair and the keyboard.

Fourth Query: “daewon song”

This is a control, as it is exactly the name of the target of the search query for this question. Both search engines provided a perfect relevancy score, which was not surprising considering that the information need was embodied in a person with a unique name, making narrowing down the relevant results quite easy based on the name alone.

1.4 Results

Table 1 shows the relevancy scores and overlap for each search engine per query.

Query	Google Relevance	DuckDuckGo Relevance	Overlap
professional skateboarder song	0.5	0.5	0.1
daewoo song	0.8	0.6	0.2
rollerblade song korea	0.0	0.0	0.0
daewon song	1.0	1.0	0.7

Table 1: Relevance Scores

Both search engines performed relatively similarly to each other for the given queries. Neither one provided a significantly higher number of relevant results for any of the queries. When given the ideal query, overlap was high, although not at 1.0, while also providing a “perfect” relevancy score of 1.0. The length of the query did not seem to be a strong controlling factor, term accuracy seemed to decide the relevancy of the top ten results for a particular topic.

2 Question 1.2

2.1 Question

Site search is another common application of search engines. In this case, search is restricted to the web pages at a given website. Compare site search to web search, vertical search, and enterprise search.

2.2 Resources

The textbook *Search Engines: Information Retrieval in Practice* [3] was used to answer this question.

2.3 Answer

Site search, as stated in the question, refers to a general search query, with results limited to pages within a particular host domain, or website. Web search is defined as a general search over all the pages the web has to offer. On one hand, site search is similar to web search because the query is unstructured. On the other, site search is a far simpler beast than a general web search. Being constrained to a particular host domain has a number of traits that make it an easier task to handle. Firstly, the number of documents is orders of magnitude smaller for site search compared to web search, allowing the search results to be extremely fresh while not representing a daunting task of crawling the entire document collection. The indexing process can also be completed much faster due to the drastically decreased number of documents. Site search never has to worry about document *spam*, items that should be excluded from any search as they are not intended to meet an information need, because the website maintaining the documents has complete control over which documents are to be indexed, leaving the decision made before the indexing process begins.

There are many examples of site-specific search features. For example, the Internet Movie Database, an informational website in the domain of film and television, provides users with a site search feature to make finding a movie, TV show, or actor a simple task. A vertical search engine may be about movies, but it is not necessarily going to have its own repository of information within the domain of interest.

The key difference between a vertical search and a site-specific search is the search agent, or who is conducting the search. With a vertical search the agent is retrieving results from many websites which are all outside the control of the vertical search engine itself. For example, Google image search returns results that are all images, but they are not all contained on one of Google's own servers. Thus, to build the search index on these documents, the vertical search provider must crawl the websites pertaining to their search topic, a task which requires care if the information is to be kept on the topic of interest.

This differs from a site-specific search in that this type search engine builds its index using documents controlled by the site administrators; the search itself is a service offered by a website to allow users more easy access to their own content. This gives the proprietors of such a service an advantage in that web crawling is not required, eliminating many problems that come along with crawling the web – non-uniformity of document models, possible network issues, storage concerns (the number of results are likely higher considering the vertical search engine must index on the entire web whereas the site specific search engine only indexes on the documents contained in that website).

Enterprise search is probably most similar to site search. This is due to the same reasons site search is dissimilar to web search and vertical search – the owner of the documents over which the search is made is also in control of the search engine itself. The owner of the enterprise search engine is likely a company with a large repository of corporate documentation, thus the need for an enterprise search solution in the first place. This allows the enterprise search provider to have the same benefits of site search described above: a smaller document collection (compared to web and vertical search) leads to faster indexing, query processing, and more fresh indexes.

3 Question 3.7

3.1 Question

Write a program that can create a valid sitemap based on the contents of a directory on your computer's hard disk. Assume that the files are accessible from a website at the URL `http://www.example.com`. For instance, if there is a file in your directory called `homework.pdf`, this would be available at `http://www.example.com/homework.pdf`. Use the real modification date on the file as the last modified time in the sitemap, and to help estimate the change frequency.

3.2 Resources

With the Python programming language [4], the script `sitemapgen.py`, in Listing 5, was created to perform the necessary tasks to complete Question 3.7. The output matches the format found at Sitemaps.org [5].

The script uses the last modified time of each file to estimate the change frequency and also escapes special characters to ensure URLs are valid. The DOM is built programatically while the script traverses the file system and when this process is complete the document is printed to standard out.

3.3 Answer

Running the script on the test directory matching the structure in Listing 1 and the output is shown in Listing 2.

```
1 smgen
2 +---sitemapgen.py
3 +---testdir
4   +---dir1
5   |   +---testfile
6   +---dir2
7       +---dir3
8       |   +---testfile
9       +---testfile
```

Listing 1: test directory structure

```
1 <?xml version="1.0" ?>
2 <urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
3   <url>
4     <loc>http://www.example.com/testdir/dir1/testfile</loc>
5     <lastmod>2016-09-18</lastmod>
6     <changefreq>daily</changefreq>
7     <priority>0.5</priority>
8   </url>
9   <url>
10    <loc>http://www.example.com/testdir/dir2/testfile</loc>
11    <lastmod>2016-09-18</lastmod>
12    <changefreq>daily</changefreq>
13    <priority>0.5</priority>
14  </url>
15  <url>
16    <loc>http://www.example.com/testdir/dir2/dir3/testfile</loc>
17    <lastmod>2016-09-18</lastmod>
18    <changefreq>daily</changefreq>
19    <priority>0.5</priority>
20  </url>
21  <url>
22    <loc>http://www.example.com/sitemapgen.py</loc>
23    <lastmod>2016-09-19</lastmod>
24    <changefreq>always</changefreq>
25    <priority>0.5</priority>
26  </url>
27 </urlset>
```

Listing 2: site map generator output

4 Question 3.8

4.1 Question

Suppose that, in an effort to crawl web pages faster, you set up two crawling machines with different starting seed URLs. Is this an effective strategy for distributed crawling? Why or why not?

4.2 Resources

The textbook *Search Engines: Information Retrieval in Practice* [3], *Parallel Crawlers* [6], and *Efficient URL Caching for World Wide Web Crawling* [7] were used to answer this question.

4.3 Answer

5 Question 3.9

5.1 Question

Write a simple single-threaded web crawler. Starting from a single input URL (perhaps a professor's web page), the crawler should download a page and then wait at least five seconds before downloading the next page. Your program should find other pages to crawl by parsing link tags found in previously crawled documents.

5.2 Resources

The Python programming language [4] was used to write the script `crawler.py` in Listing 4 to crawl the web. Some sample output can be found in Listing 3.

5.3 Answer

```
1 retrieving http://cs.odu.edu/~mln
2 retrieving http://www.cs.odu.edu/~mln/
3 retrieving http://www.cs.odu.edu/
4 retrieving http://www.odu.edu
5 retrieving http://www.cs.odu.edu/~mln/research/
6 retrieving http://www.cs.odu.edu/~mln/pubs/
7 retrieving http://www.cs.odu.edu/~mln/teaching/
8 retrieving http://www.cs.odu.edu/~mln/service/
9 retrieving http://www.cs.odu.edu/~mln/personal/
10 retrieving http://www.larc.nasa.gov/
11 retrieving http://sils.unc.edu/
12 retrieving http://www.openarchives.org/pmh/
13 retrieving http://www.openarchives.org/ore/
14 retrieving http://www.mementoweb.org/guide/rfc/ID/
15 retrieving http://www.openarchives.org/rs/toc
16 retrieving http://ntrs.nasa.gov/
17 retrieving http://www.nsf.gov/awardsearch/showAward.do?AwardNumber=0643784
```

Listing 3: crawler.py output

6 Appendix A

```
1 import argparse
2 import requests
3 from time import sleep
4 from collections import deque
5 from bs4 import BeautifulSoup
6
7
8 def crawl():
9     while True:
10         url = frontier.popleft()
11         if url not in visited:
12             visited.add(url)
13             print 'retrieving', url
14             try:
15                 res = requests.get(url)
16             except:
17                 continue
18             if res.ok:
19                 soup = BeautifulSoup(res.text, 'html.parser')
20                 links = soup.find_all('a')
21                 for link in links:
22                     try:
23                         linkurl = link['href'].strip()
24                         if linkurl not in visited:
25                             frontier.append(linkurl)
26                     except KeyError:
27                         continue
28         sleep(5)
29
30
31 if __name__ == '__main__':
32     parser = argparse.ArgumentParser('web crawler')
33     parser.add_argument('--host', default='http://cs.odu.edu/~mln')
34     args = parser.parse_args()
35     visited = set()
36     frontier = deque()
37     frontier.append(args.host)
38     crawl()
```

Listing 4: crawler.py

```

1 import sys
2 import os
3 import argparse
4 import datetime
5 import time
6 import urllib
7
8 from os.path import getmtime, isdir, isfile
9
10 import xml.dom.minidom as md
11
12
13 def append(doc, urlset, loc, lastmod, changefreq, priority='0.5'):
14     url = doc.createElement('url')
15     urlset.appendChild(url)
16
17     loc_element = doc.createElement('loc')
18     loc_element.appendChild(doc.createTextNode(loc))
19     url.appendChild(loc_element)
20
21     lastmod_element = doc.createElement('lastmod')
22     lastmod_element.appendChild(doc.createTextNode(lastmod))
23     url.appendChild(lastmod_element)
24
25     changefreq_element = doc.createElement('changefreq')
26     changefreq_element.appendChild(doc.createTextNode(changefreq))
27     url.appendChild(changefreq_element)
28
29     priority_element = doc.createElement('priority')
30     priority_element.appendChild(doc.createTextNode(priority))
31     url.appendChild(priority_element)
32
33
34 YEAR = 3.154e+7
35 MONTH = 2.592e+6
36 WEEK = 604800.0
37 DAY = 86400
38 HOUR = 3600
39 MINUTE = 60
40
41 def estimate_changefreq(posixtime):
42     timenow = time.time()
43     delta = timenow - posixtime
44     if delta > YEAR:
45         return 'never'
46     elif delta > MONTH:
47         return 'yearly'
48     elif delta > WEEK:
49         return 'monthly'
50     elif delta > DAY:
51         return 'weekly'
52     elif delta > HOUR:
53         return 'daily'
54     elif delta > MINUTE:
55         return 'hourly'
56     else:
57         return 'always'
58
59
60 def convertdate(posixtime):
61     return datetime.datetime.utcfromtimestamp(posixtime).strftime('%Y-%m-%d')
62
63
64 def delve(root, folder, doc, urlset):
65     items = os.listdir(root + folder)
66     for item in items:
67         filepath = root + folder + item
68         if isfile(filepath):
69             loc = args.host + urllib.quote(folder + item)
70             lastmodsecs = getmtime(filepath)
71             lastmod = convertdate(lastmodsecs)
72             changefreq = estimate_changefreq(lastmodsecs)
73             append(doc, urlset, loc, lastmod, changefreq)
74         elif isdir(filepath):
75             delve(root, folder + item + os.sep, doc, urlset)
76

```

```

77
78 def test(doc, urlset):
79     append(doc, urlset, 'www.google.com', '2016-09-17', 'always', '0.8')
80     append(doc, urlset, 'www.duckduckgo.com', '2016-09-17', 'daily')
81     print doc.toprettyxml()
82
83
84 if __name__ == '__main__':
85     # parse arguments
86     parser = argparse.ArgumentParser('site map generator')
87     parser.add_argument(
88         '-test',
89         '-t',
90         action='store_true')
91     parser.add_argument(
92         '--path',
93         '-p',
94         default='.')
95     parser.add_argument(
96         '--host',
97         default='http://www.example.com/')
98     args = parser.parse_args()
99
100     # create a document
101     doc = md.Document()
102     urlset = doc.createElement('urlset')
103     urlset.setAttribute('xmlns', 'http://www.sitemaps.org/schemas/sitemap/0.9')
104     doc.appendChild(urlset)
105
106     # if desired, perform simple test and return
107     if args.test:
108         test(doc, urlset)
109         sys.exit(0)
110
111     # parse path from args
112     path = args.path
113     if path[len(path)-1] != os.sep:
114         path = path + os.sep
115
116     # iterate over all items in doc
117     delve(path, '', doc, urlset)
118
119     print doc.toprettyxml()

```

Listing 5: sitemapgen.py

7 References

- [1] Google. Available at: <http://www.google.com>. Accessed: 2016/09/17.
- [2] DuckDuckGo. Available at: <http://www.duckduckgo.com>. Accessed: 2016/09/17.
- [3] Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Information Retrieval in Practice*. Pearson, first edition, February 2009.
- [4] The Python Programming Language. Available at: <https://www.python.org/>. Accessed: 2016/09/17.
- [5] Sitemaps XML format. Available at: <http://www.sitemaps.org/protocol.html>. Accessed: 2016/09/17.
- [6] Junghoo Cho and Hector Garcia-Molina. Parallel Crawlers. In *Proceedings of the 11th International Conference on World Wide Web*, WWW '02, pages 124–135, New York, NY, USA, 2002. ACM.
- [7] Andrei Z. Broder, Marc Najork, and Janet L. Wiener. Efficient URL Caching for World Wide Web Crawling. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 679–689, New York, NY, USA, 2003. ACM.