Assignment 1

CS834 Introduction to Information Retrieval
Fall 2017

Polykarpos Thomadakis

September 20, 2017

Question 1.1

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?

Answer

Bing and DuckDuckGo were the two search engines that I compared for the purposes of this assignment. I searched for something that I expected to be a bit hard for them to return the best results without enough words. My search was for examples of Message Passing Interface (MPI) code in C language.
The precision is defined as the number of relevant results recalled by the total number of results recalled :

$$Precision = \frac{Relevant\ results}{Total\ number\ of\ results}$$

The overlap is defined as the number of results that appear in both search engines for the same query:

$$Overlap = \frac{Results \in \mathcal{D} \cap \mathcal{B}}{Number\ \ of\ results\ per\ search\ engine}$$

where $\mathcal{D}, \mathcal{B}$ are the sets of results for DuckDuckGo and Bing respectively.

Note for DuckDuckGo: I did not include the result that is an advertisement in my evaluations,since advertisements are usually biased to come up even when they might not be that relevant.

I started with a one word query, simply MPI:
The acronym MPI is used for many different things, so none of the engines did that well with the one-word query.Bing did not return any relevant result at all, as opposed to DuckDuckGo which returned 3.

As for the overlap, it was zero for the relevant results, however 3 out of the non relevant results are common. I think it is important to note that and also the fact that the top result for both engines was the same.
The results can be seen in figure 1 and the precision in each case is:

$$Precision_{DuckDuckGo} = \frac{3}{10} = 0.3$$

$$Precision_{Bing} = \frac{0}{10} = 0$$

$$Overlap_{Relevant} = \frac{0}{10} = 0 \qquad Overlap_{total} = \frac{3}{10} = 0.3$$

The next query was with 2 words, MPI C:
This time the engines did a much better job.Seven of the results were relevant for DuckDuckGo with the top 4 being relevant and two of the three irrelevants at the last two positions. Bing came short with 6 relevant results, however it made a big difference compared to the previous attempt which had no relevant results. The third, the seventh, the nineth and the tenth results were irrelevant.
Six of the relevant results appear in both search engines and nine of the total results are common.
The results are presented in figure 2.

$$Precision_{DuckDuckGo} = \frac{7}{10} = 0.7$$

$$Precision_{Bing} = \frac{6}{10} = 0.6$$

$$Overlap_{Relevant} = \frac{6}{10} = 0.6 \qquad Overlap_{total} = \frac{9}{10} = 0.9$$

For the last attempt I used 3 words, MPI C examples:
In this case both engines did their best, returning only relevant results. Seven of the results are common for this case with the top two results being in the same ranking order. Another interesting finding is that the total overlap of the two engines has decreased even thought their precision has increased.
The results are presented in figure 3.

$$Precision_{DuckDuckGo} = \frac{10}{10} = 1.0$$

$$Precision_{Bing} = \frac{10}{10} = 1.0$$

$$Overlap_{Relevant} = \frac{7}{10} = 0.7 \qquad Overlap_{total} = \frac{7}{10} = 0.7$$

The overall results can be seen in table 1. It can be easily noted that the DuckDuckGo search engine outperforms Bing in all cases. In addtion to that, we can see that increasing the length of the query results in a better precision.

	$Prec_{DuckDuckGo}$	$Prec_{Bing}$	Relevant Overlap	Total Overlap
MPI	0.3	0	0	0.3
MPI C	0.7	0.6	0.6	0.9
MPI C examples	1.0	1.0	0.7	0.7

Table 1: Overall results

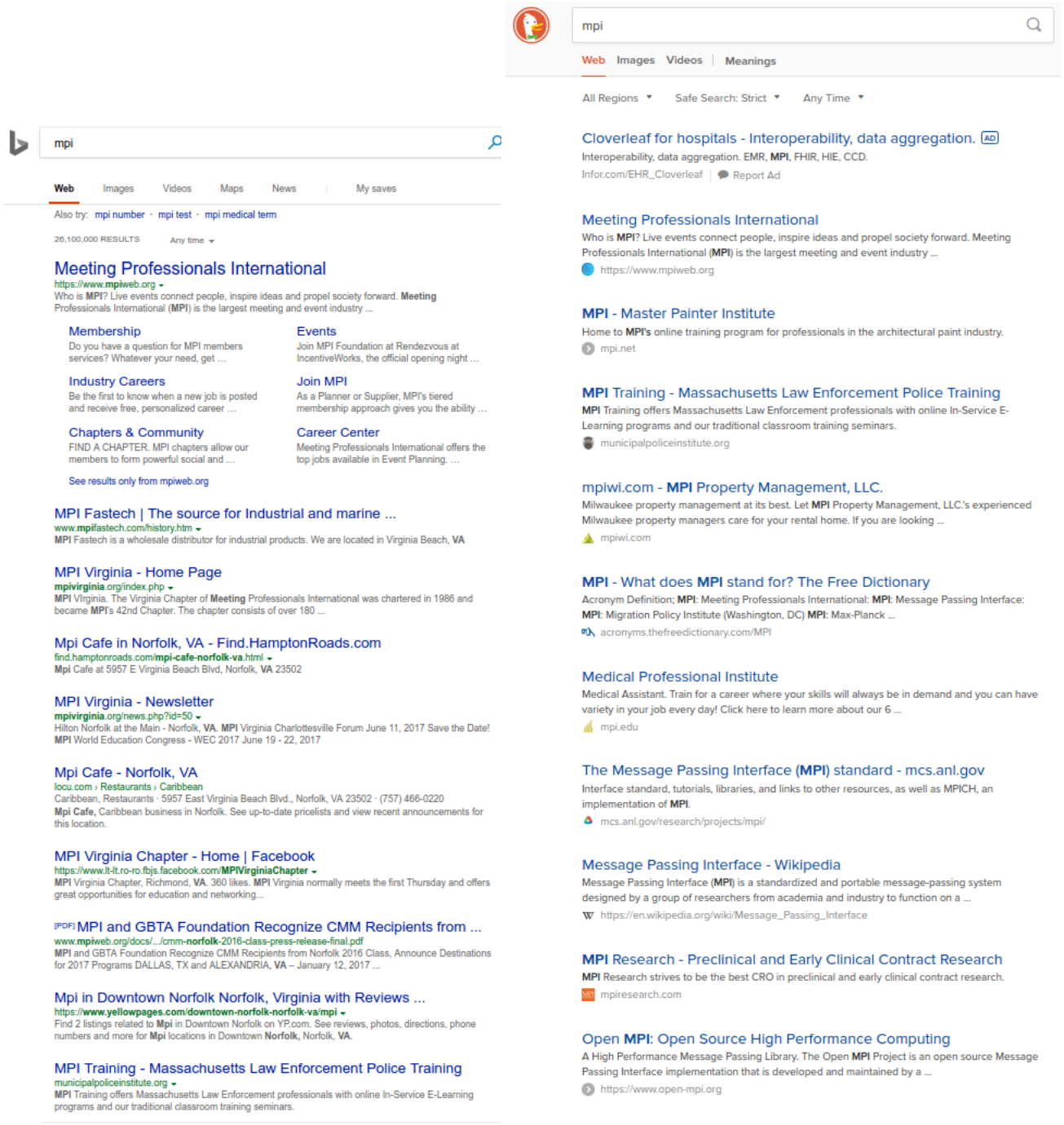

Figure 1: Comparison of the results for query: MPI

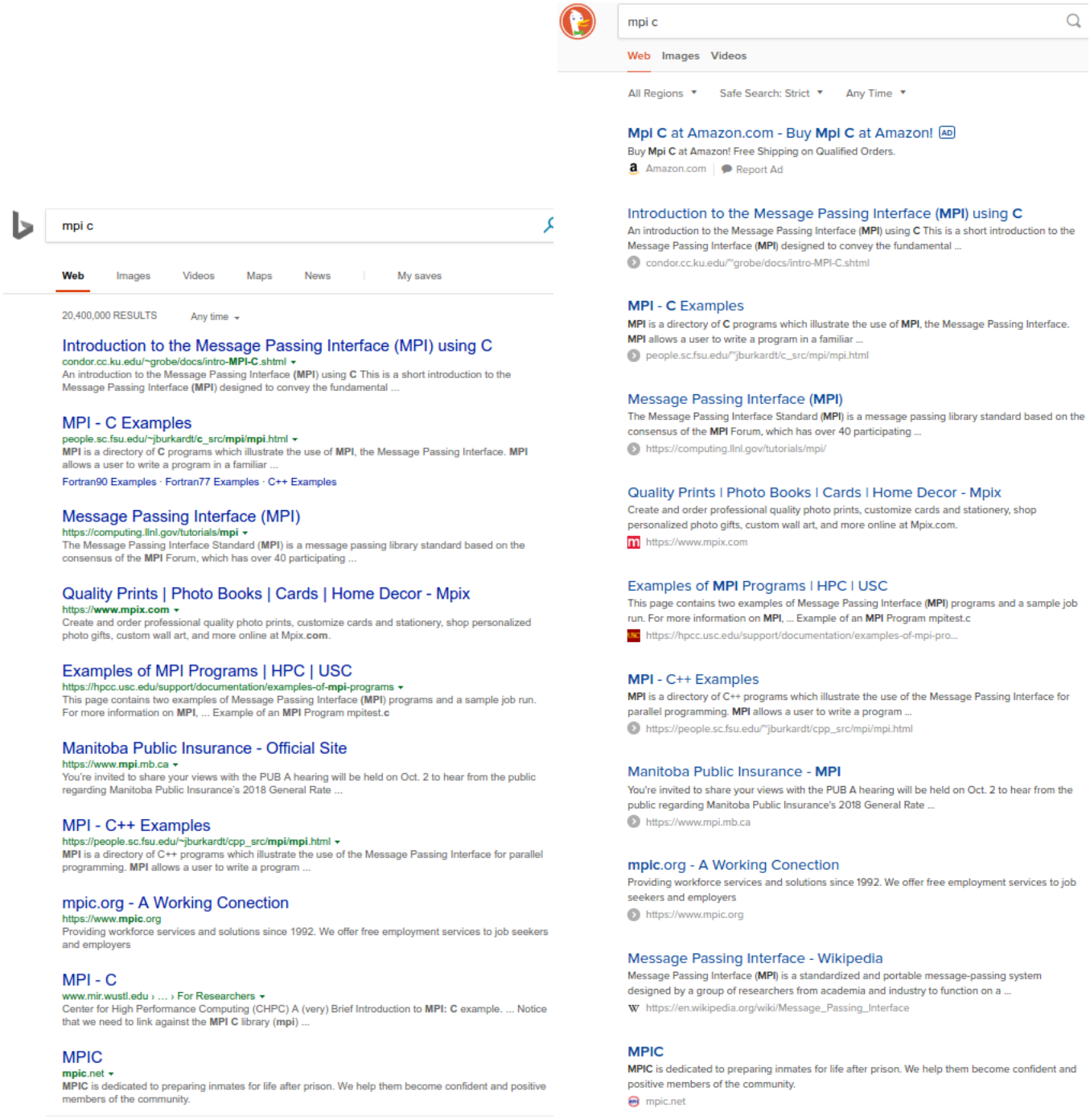

Figure 2: Comparison of the results for query: MPI C

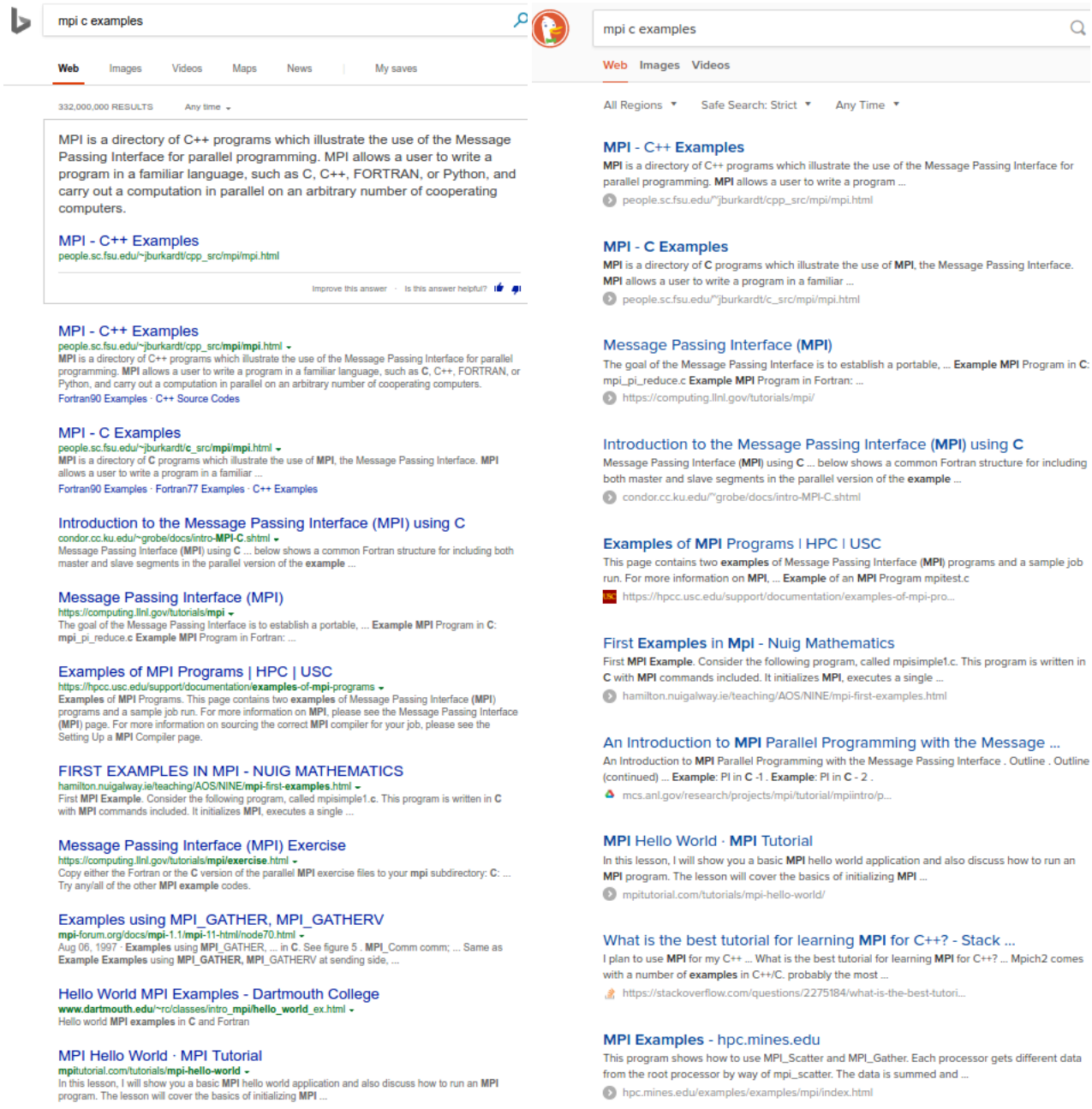

Figure 3: Comparison of the results for query: MPI C examples

Question 1.4

List five web services or sites that you use that appear to use search, not including web search engines. Describe the role of search for that service. Also describe whether the search is based on a database or grep style of matching, or if the search is using some type of ranking.

Answer

Here are five sites that I use that appear to use search:

1. **imdb.com**
   The Internet Movie Database (abbreviated IMDb) is an online database of information related to films, television programs and video games, including cast, production crew, fictional characters, biographies, plot summaries, trivia and reviews. The user uses a search query to get information about movies, series or games that are relevant. The search looks to be done against a database.

2. **stackoverflow.com**
   Stack Overflow is a website featuring questions and answers on a wide range of topics in computer programming. The website serves as a platform for users to ask and answer questions, and, through membership and active participation, to upvote or downvote questions and answers based on their content. This site seems to use a grep style search and also features ranking based on the upvotes of the relevant posts.

3. **play.google.com**
   Google Play is a digital distribution service. It serves as the official app store for the Android operating system, allowing users to browse and download applications developed with the Android software development kit (SDK) and published through Google. Google Play also serves as a digital media store, offering music, magazines, books, movies, and television programs. Google Play seems to use a database search and using ranking based on the popularity of the results and their user provided reviews.

4. **craigslist.com**
   Craigslist is a classified advertisements website with sections devoted to jobs, housing, personals, for sale, items wanted, services, community, gigs, resumes, and discussion forums. A user can choose a section and make a search for results relevant to it. The site seems to be using a database search style.

5. **github.com**
   GitHub is a web-based Git or version control repository and Internet hosting service. It is mostly used for code. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project. Github seems to use a database style of search and offers optional ranking based on best match, highest rating, freshness, most forked.

Question 3.7

Write a program that can create a valid sitemap based on the contents of a directory on your computers 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.

Answer

The program discovers all the files in the directory given from the user as a command line argument. Then it continues with the files and subdirectories of its subdirectories and goes on until all the files and directories under the given directory have been discovered recursively. To achieve this I use the os.walk() function from the os directory.
For each file found an xml url structure is generated where the location is registered and then the last modification date information is retrieved with the os.stat() function and is included in the url structure of the file.

Once all files in the tree under the origin directory have been discovered the procedure ends and the xml file is generated with the appropriate format and the structures generated are written on it. The user can specify the path where the file will be created, the default location is the current directory. The source code used can be seen in listing 1 and the sitemap generated for the directory of the assignment can be found in listing 2.

```
import os
import time
import sys
import argparse

parser = argparse.ArgumentParser('smapgen')
parser.add_argument('path',help='Path to the directory that will be used to generate sitemap')
parser.add_argument('--path_to_gen','-p',help='Path where the generated sitemap.xml will be put [default= Current directory]',default='./')
args = parser.parse_args();
p_to_gen = args.path_to_gen+'sitemap.xml';
root_dir = args.path

out=''
out+='<?xml version="1.0" encoding="UTF-8"?>\n'
out+='<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">\n'
for root,dirs,files in os.walk(root_dir):
  for f in files:
    loc = root+'/'+f
    url = root.replace(root_dir,"http://www.example.com")
    out+='  <url>\n'
    out+= '    <loc>'+url+'/'+f+'</loc>\n'
    out+= '    <lastmod>'+time.strftime("%Y-%m-%d",time.localtime(os.path.getmtime(loc)))+'</lastmod>\n'
    out+= '  </url>\n'
out+= '</urlset>'

with open(p_to_gen,'w') as f:
  f.write(out)
  f.close()
```

Listing 1: Sitemap generator

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.example.com/A1.pdf</loc>
    <lastmod>2017-09-03</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/compress.py</loc>
    <lastmod>2017-09-03</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/sample_output</loc>
```

```
    <lastmod>2017-09-02</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/A1.synctex.gz</loc>
    <lastmod>2017-09-03</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/crawl.py</loc>
    <lastmod>2017-09-02</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/A1.log</loc>
    <lastmod>2017-09-03</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/smapgen.py</loc>
    <lastmod>2017-09-03</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/A1.tex</loc>
    <lastmod>2017-09-03</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/decompress.py</loc>
    <lastmod>2017-09-03</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/A1.aux</loc>
    <lastmod>2017-09-03</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/sitemap.xml</loc>
    <lastmod>2017-09-03</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/Wiki.html</loc>
    <lastmod>2017-09-03</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/InformationRetrieval.pdf</loc>
    <lastmod>2017-09-02</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/Wc.html</loc>
    <lastmod>2017-09-03</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/NewFOlder/test</loc>
    <lastmod>2017-09-03</lastmod>
  </url>
  <url>
    <loc>http://www.example.com/NewFOlder/Another/anothertest</loc>
    <lastmod>2017-09-03</lastmod>
  </url>
</urlset>
```

Listing 2: Generated sitemap

Question 3.9

Write a simple single-threaded web crawler. Starting from a single input URL (perhaps a professors 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.

Answer

The crawler starts with one page on its list, the one provided by the user or the default. When all links of the page have been found, the list is filled with the new pages to be crawled. The crawler continues crawling pages in this list, parsing their links.
Once all of the pages in the list have been parsed it will continue with their respective links and so on. In order to keep the running time reasonable the user can set the depth in which this procedure will continue crawling. The default depth is 2.

To avoid downloading big irrelevant files, the crawler will skip non-HTML files by checking the content-type in the header. Command line arguments can be used to set the path where the crawler will create a directory and store the downloaded files.
The source code for the crawler is presented in listing 3 and a sample of the output for depth 2 can be seen in listing 4.

```
from bs4 import BeautifulSoup
from urlparse import urljoin
import argparse
import requests
from time import sleep
import os

def crawl(depth,s_url):
  i = 1
  file_num = 0
  s_url = s_url.strip()
  print 'URL to start crawling:',s_url
  print 'Maximum depth:',depth
  urls = [s_url]
  while i<=depth:
    deeper_links = list()
    for url in urls:
      if url not in checked_urls:
        try:
          # Request the link from the Internet and set a timeout to prevent
          # hanging
          request = requests.get(url,stream=True,timeout=1)
          # Will only parse html files, skips any other format
          if( 'text/html' not in request.headers['Content-Type'] ):
            print 'Skipping ',url
            continue
          print 'Crawling ',url,
        except :
          continue
        # Check if link is valid or not
        if(request.status_code==200):
          checked_urls.add(url)
          print ' URL OK'
        else:
          print ' URL NOT OK'
          continue
        file_num +=1
        with open(path+'/'+str(file_num)+'.html','w') as file:
          file.write(request.text.encode('utf-8'))
          file.close()
        soup = BeautifulSoup(request.text,'html.parser')
        # Retrieve the links in the html file
        links = soup.find_all('a')
        for link in links:
```

```
            new_link = link.get('href')
            if  new_link :
              new_link = new_link.strip()
              # Make link absolute if needed
              new_link = urljoin(url,new_link)
              # If link not crawled yet add to pending links
              if new_link not in checked_urls:
                deeper_links.append(new_link)
        # Wait 5 seconds before downloading the next html file
        sleep(5)
    print len(deeper_links),'links for next depth'
    urls = deeper_links
    print 'Finished depth ',i
    i=i+1

# Set the command line arguments
parser = argparse.ArgumentParser('Crawler Parser')
parser.add_argument('--url','-u',help='The url to start crawling from [default= http
    ://www.cs.odu.edu/~mln/ ',default='http://www.cs.odu.edu/~mln/')
parser.add_argument('--depth','-d',help='The crawl depth [default= 5]',default=5,type
    =int)
parser.add_argument('--path','-p',help='The path where the crawl directory will be
    created to put the downloaded files [default= current directory]',default='.')
args = parser.parse_args();
depth = args.depth
s_url = args.url
path = os.path.abspath(args.path)+'/crawl'
if not os.path.exists(path):
    os.makedirs(path)
print 'Files will be downloaded at',path
# A set of all previously met links
checked_urls = set()
crawl(depth,s_url)
print 'Crawler finished, you can find the downloaded files at',path
```

Listing 3: Web crawler

```
Files will be downloaded at /home/poll/Desktop/crawl
URL to start crawling: http://www.cs.odu.edu/~mln/
Maximum depth: 2
Crawling  http://www.cs.odu.edu/~mln/  URL OK
24 links for next depth
Finished depth  1
Crawling  http://www.cs.odu.edu/  URL OK
Crawling  http://www.odu.edu  URL OK
Crawling  http://www.cs.odu.edu/~mln/research/  URL OK
Crawling  http://www.cs.odu.edu/~mln/pubs/  URL OK
Crawling  http://www.cs.odu.edu/~mln/teaching/  URL OK
Crawling  http://www.cs.odu.edu/~mln/service/  URL OK
Crawling  http://www.cs.odu.edu/~mln/personal/  URL OK
Crawling  http://www.larc.nasa.gov/  URL OK
Crawling  http://sils.unc.edu/  URL OK
Crawling  http://www.openarchives.org/pmh/  URL OK
Crawling  http://www.openarchives.org/ore/  URL OK
Crawling  http://www.mementoweb.org/guide/rfc/ID/  URL OK
Crawling  http://www.openarchives.org/rs/toc  URL OK
Crawling  http://ntrs.nasa.gov/  URL OK
Crawling  http://www.nsf.gov/awardsearch/showAward.do?AwardNumber=0643784  URL OK
Skipping  http://www.cs.odu.edu/~mln/cv.pdf
Skipping  http://www.cs.odu.edu/~mln/nsf-cv-2014.pdf
Crawling  http://www.cs.odu.edu/~mln/lineage.html  URL OK
Crawling  http://www.cs.odu.edu/~mln/travel.html  URL OK
Skipping  http://www.cs.odu.edu/~mln/mln-ad.pdf
Crawling  https://storify.com/michaelnelson/coverage-of-ws-dl-members-and-research
    URL OK
Crawling  http://ws-dl.blogspot.com/  URL OK
```

```
Crawling  http://twitter.com/phonedude_mln  URL OK
Crawling  https://twitter.com/phonedude_mln  URL OK
2973 links for next depth
Finished depth  2
Crawler finished, you can find the downloaded files at /home/poll/Desktop/crawl
```

Listing 4: Web crawler sample output

Question 3.12

Design a compression algorithm that compresses HTML tags. Your algorithm should detect tags in an HTML file and replace them with a code of your own design that is smaller than the tag itself. Write an encoder and decoder program.

Answer

To create the compression algorithm I introduced a 1-1 mapping between each html tag and my compressed tags. HTML tags with 3 or more letters are compressed to 2-3 at most. I start by loading the HTML file and finding all HTML tags in it using regular expressions to match them. Once a tag is matched it will be checked for transformation using the mapping introduced.
For the decompression algorithm I follow the same procedure, but now I perform the mapping the reverse way i.e map the custom tags to the original ones and transform them when needed.
The user must specify an input and output file name for both programs. Of course the output file should not exist, or it will be overwritten.The source code for both programs is found in listings 5 and 6 respectively. Sample input, output can be found in listings 7 and 8. Also the full files are available on my github repository but are not included here because of their size.

```
import re
import argparse

def repl(matchobj):
  for k,v in mapping.iteritems():
    if matchobj.group() == '<'+k :
      return '<'+v
    elif matchobj.group() == '</'+k :
      return '</'+v
  return matchobj.group()


mapping = {'!DOCTYPE':'!DT','abbr':'ab','acronym':'ac','address':'ad','applet':'ap','area':'ar','article':'art'
,'aside':'as','audio':'au','basefont':'bf','base':'ba','bdi':'bi','bdo':'bo','big':'bg','blockquote':'bq','body':'bd',
'button':'bu','canvas':'cv','caption':'ca','center':'ce','cite':'ci','code':'cd','colgroup':'cg','datalist':'da','del':'de',
'details':'da','dfn':'df','dialog':'dg','dir':'dr','div':'dv','embed':'ed','fieldset':'fs','figure':'fi','font':'ft','footer':'fr',
'form':'fm','frame':'fe','frameset':'fa','header':'he','head':'hd','html':'hl','iframe':'ir','img':'im','input':'ip','ins':'in',
'kbd':'kb','keygen':'kg','label':'la','legend':'le','link':'ln','main':'mi','map':'mp','mark':'mk','menuitem':'mm','menu':'mn',
'meta':'mt','meter':'mr','nav':'na','noframes':'nf','nscript':'ns','object':'ob','optgroup':'og','option':'op','output':'ou',
'param':'pa','picture':'pi','pre':'pr','progress':'pg','ruby':'rb','samp':'sa','script':'sc','section':'se','select':'sl',
'small':'sm','source':'so','span':'sp','strike':'st','strong':'sn','style':'sy','sub':'su','summary':'sr','table':'ta','tbody':'tb',
'textarea':'ta','tfoot':'tf','thead':'te','time':'ti','title':'tl','track':'tk','var':'vr','video':'vd','wbr':'wb'}
```

```
23
parser = argparse.ArgumentParser('compress')
25 parser.add_argument('path',help='Path to the file that will be compressed')
parser.add_argument('compressed',help='Path to the compressed file that will be
    generated')
27 args = parser.parse_args();
p_comp = args.compressed
29 p_path = args.path

31 f = open(p_path,'r')
out=""
33
for line in f:
35   line = re.sub('</?[A-Za-z0-9]+',repl,line)
   out+=line
37 f.close()
f = open(p_comp,'w')
39 f.write(out)
f.close()
```

Listing 5: Compress source

```
import re
2 import argparse

4 def revrepl(matchobj):
   for k,v in mapping.iteritems():
6     if matchobj.group() == '<'+v :
       return '<'+k
8     elif matchobj.group() == '</'+v :
       return '</'+k
10   return matchobj.group()


12
mapping = {'!DOCTYPE':'!DT','abbr':'ab','acronym':'ac','address':'ad','applet':'ap','
    area':'ar','article':'art'
14 ,'aside':'as','audio':'au','basefont':'bf','base':'ba','bdi':'bi','bdo':'bo','big':'
    bg','blockquote':'bq','body':'bd',
'button':'bu','canvas':'cv','caption':'ca','center':'ce','cite':'ci','code':'cd','
    colgroup':'cg','datalist':'da','del':'de',
16 'details':'da','dfn':'df','dialog':'dg','dir':'dr','div':'dv','embed':'ed','fieldset'
    :'fs','figure':'fi','font':'ft','footer':'fr',
'form':'fm','frame':'fe','frameset':'fa','header':'he','head':'hd','html':'hl','
    iframe':'ir','img':'im','input':'ip','ins':'in',
18 'kbd':'kb','keygen':'kg','label':'la','legend':'le','link':'ln','main':'mi','map':'mp
    ','mark':'mk','menuitem':'mm','menu':'mn',
'meta':'mt','meter':'mr','nav':'na','noframes':'nf','nscript':'ns','object':'ob','
    optgroup':'og','option':'op','output':'ou',
20 'param':'pa','picture':'pi','pre':'pr','progress':'pg','ruby':'rb','samp':'sa','
    script':'sc','section':'se','select':'sl',
'small':'sm','source':'so','span':'sp','strike':'st','strong':'sn','style':'sy','sub'
    :'su','summary':'sr','table':'ta','tbody':'tb',
22 'textarea':'ta','tfoot':'tf','thead':'te','time':'ti','title':'tl','track':'tk','var'
    :'vr','video':'vd','wbr':'wb'}
nout=""
24
parser = argparse.ArgumentParser('decompress')
26 parser.add_argument('compressed',help='Path to the compressed file')
parser.add_argument('decompressed',help='Path to the decompressed file that will be
    generated')
28 args = parser.parse_args();
p_comp = args.compressed
30 p_decomp = args.decompressed
f = open(p_comp,'r')
32 for line in f:
   line = re.sub('</?[A-Za-z0-9]+',revrepl,line)
```

```
    nout+=line

f.close()
f = open(p_decomp,'w')
f.write(nout)
f.close()
```

Listing 6: Decompress source

```
<!DOCTYPE html>
<html class="client-nojs" lang="en" dir="ltr">
<head>
<meta charset="UTF-8"/>
<title>Wikipedia, the free encyclopedia</title>
<script>document.documentElement.className = document.documentElement.className.replace( /(^|\s)client-nojs(\s|$)/, "$1client-js$2" );</script>
<link rel="stylesheet" href="/w/load.php?debug=false&amp;lang=en&amp;modules=ext.uls.interlanguage%7Cext.visualEditor.desktopArticleTarget.noscript%7Cext.wikimediaBadges%7Cmediawiki.legacy.commonPrint%2Cshared%7Cmediawiki.sectionAnchor%7Cmediawiki.skinning.interface%7Cskins.vector.styles&amp;only=styles&amp;skin=vector"/>
<script async="" src="/w/load.php?debug=false&amp;lang=en&amp;modules=startup&amp;only=scripts&amp;skin=vector"></script>
<meta name="ResourceLoaderDynamicStyles" content=""/>
<link rel="stylesheet" href="/w/load.php?debug=false&amp;lang=en&amp;modules=ext.gadget.charinsert-styles&amp;only=styles&amp;skin=vector"/>
<link rel="stylesheet" href="/w/load.php?debug=false&amp;lang=en&amp;modules=site.styles&amp;only=styles&amp;skin=vector"/>
<meta name="generator" content="MediaWiki 1.30.0-wmf.16"/>
<meta name="referrer" content="origin-when-cross-origin"/>
<meta property="og:image" content="https://upload.wikimedia.org/wikipedia/commons/thumb/9/94/Manon_Williams_Croeso_Cymru.jpg/1200px-Manon_Williams_Croeso_Cymru.jpg"/>
<link rel="alternate" href="android-app://org.wikipedia/http/en.m.wikipedia.org/wiki/Main_Page"/>
<link rel="alternate" type="application/atom+xml" title="Wikipedia picture of the day feed" href="/w/api.php?action=featuredfeed&amp;feed=potd&amp;feedformat=atom"/>
<link rel="alternate" type="application/atom+xml" title="Wikipedia featured articles feed" href="/w/api.php?action=featuredfeed&amp;feed=featured&amp;feedformat=atom"/>
<link rel="alternate" type="application/atom+xml" title="Wikipedia &amp;quot;On this day ...&amp;quot; feed" href="/w/api.php?action=featuredfeed&amp;feed=onthisday&amp;feedformat=atom"/>
<link rel="apple-touch-icon" href="/static/apple-touch/wikipedia.png"/>
<link rel="shortcut icon" href="/static/favicon/wikipedia.ico"/>
<link rel="search" type="application/opensearchdescription+xml" href="/w/opensearch_desc.php" title="Wikipedia (en)"/>
<link rel="EditURI" type="application/rsd+xml" href="//en.wikipedia.org/w/api.php?action=rsd"/>
<link rel="license" href="//creativecommons.org/licenses/by-sa/3.0/"/>
<link rel="canonical" href="https://en.wikipedia.org/wiki/Main_Page"/>
<link rel="dns-prefetch" href="//login.wikimedia.org"/>
<link rel="dns-prefetch" href="//meta.wikimedia.org" />
<!--[if lt IE 9]><script src="/resources/lib/html5shiv/html5shiv.min.js"></script><![endif]-->
</head>
<body class="mediawiki ltr sitedir-ltr mw-hide-empty-elt ns-0 ns-subject page-Main_Page rootpage-Main_Page skin-vector action-view">    <div id="mw-page-base" class="noprint"></div>
```

Listing 7: Sample of the original Wikipedia source file

```
<!DOCTYPE html>
```

```
<hl class="client-nojs" lang="en" dir="ltr">
<hd>
<mt charset="UTF-8"/>
<tl>Wikipedia, the free encyclopedia</tl>
<sc>document.documentElement.className = document.documentElement.className.replace(
    /(^|\s)client-nojs(\s|$)/, "$1client-js$2" );</sc>
<ln rel="stylesheet" href="/w/load.php?debug=false&amp;lang=en&amp;modules=ext.uls.
    interlanguage%7Cext.visualEditor.desktopArticleTarget.noscript%7Cext.
    wikimediaBadges%7Cmediawiki.legacy.commonPrint%2Cshared%7Cmediawiki.sectionAnchor
    %7Cmediawiki.skinning.interface%7Cskins.vector.styles&amp;only=styles&amp;skin=
    vector"/>
<sc async="" src="/w/load.php?debug=false&amp;lang=en&amp;modules=startup&amp;only=
    scripts&amp;skin=vector"></sc>
<mt name="ResourceLoaderDynamicStyles" content=""/>
<ln rel="stylesheet" href="/w/load.php?debug=false&amp;lang=en&amp;modules=ext.gadget
    .charinsert-styles&amp;only=styles&amp;skin=vector"/>
<ln rel="stylesheet" href="/w/load.php?debug=false&amp;lang=en&amp;modules=site.
    styles&amp;only=styles&amp;skin=vector"/>
<mt name="generator" content="MediaWiki 1.30.0-wmf.16"/>
<mt name="referrer" content="origin-when-cross-origin"/>
<mt property="og:image" content="https://upload.wikimedia.org/wikipedia/commons/thumb
    /9/94/Manon_Williams_Croeso_Cymru.jpg/1200px-Manon_Williams_Croeso_Cymru.jpg"/>
<ln rel="alternate" href="android-app://org.wikipedia/http/en.m.wikipedia.org/wiki/
    Main_Page"/>
<ln rel="alternate" type="application/atom+xml" title="Wikipedia picture of the day
    feed" href="/w/api.php?action=featuredfeed&amp;feed=potd&amp;feedformat=atom"/>
<ln rel="alternate" type="application/atom+xml" title="Wikipedia featured articles
    feed" href="/w/api.php?action=featuredfeed&amp;feed=featured&amp;feedformat=atom"
    />
<ln rel="alternate" type="application/atom+xml" title="Wikipedia &quot;On this day
    ...&quot; feed" href="/w/api.php?action=featuredfeed&amp;feed=onthisday&amp;
    feedformat=atom"/>
<ln rel="apple-touch-icon" href="/static/apple-touch/wikipedia.png"/>
<ln rel="shortcut icon" href="/static/favicon/wikipedia.ico"/>
<ln rel="search" type="application/opensearchdescription+xml" href="/w/
    opensearch_desc.php" title="Wikipedia (en)"/>
<ln rel="EditURI" type="application/rsd+xml" href="//en.wikipedia.org/w/api.php?
    action=rsd"/>
<ln rel="license" href="//creativecommons.org/licenses/by-sa/3.0/"/>
<ln rel="canonical" href="https://en.wikipedia.org/wiki/Main_Page"/>
<ln rel="dns-prefetch" href="//login.wikimedia.org"/>
<ln rel="dns-prefetch" href="//meta.wikimedia.org" />
<!--[if lt IE 9]><sc src="/resources/lib/html5shiv/html5shiv.min.js"></sc><![endif]--
    >
</hd>
<bd class="mediawiki ltr sitedir-ltr mw-hide-empty-elt ns-0 ns-subject page-Main_Page
     rootpage-Main_Page skin-vector action-view">    <dv id="mw-page-base" class="
    noprint"></dv>
```

Listing 8: Sample of the compressed Wikipedia source file