1 Web Crawling and Web Search

A web crawler is a program that systematically visits pages on the internet. For programming assignment 4, you should write a web crawler that builds a database of the words stored on the pages that are visited. The database should be a Python dictionary whose keys are the absolute URLs of the sites visited and whose values are lists of the words on the sites visited.

After building the URL–word-list dictionary, your program should build a reverse index of the database. This is another dictionary whose keys are the words in the original dictionary and whose values are the URLs. For example, if the first dictionary is

```python
{'http://cs.usfca.edu/~peter/cs110/hello.html':
 ['hello',
  'ciao',
  'goodday'],
'http://cs.usfca.edu/~peter/cs110/goodbye.html':
 ['goodbye',
  'bye',
  'ciao',
  'goodday'],
}
```

Then the reverse index would be

```python
{'hello':
 ['http://cs.usfca.edu/~peter/cs110/hello.html'],
'ciao':
 ['http://cs.usfca.edu/~peter/cs110/hello.html',
  'http://cs.usfca.edu/~peter/cs110/goodbye.html'],
'goodday':
 ['http://cs.usfca.edu/~peter/cs110/hello.html',
  'http://cs.usfca.edu/~peter/cs110/goodbye.html'],
'goodbye':
 ['http://cs.usfca.edu/~peter/cs110/goodbye.html'],
'bye':
 ['http://cs.usfca.edu/~peter/cs110/goodbye.html']}
```

After building the two dictionaries, your program should give the user a choice of commands:

- Search the database for a word (s)
- Open a URL in a browser (o)
Print the dictionary with keys that are URLs (u)
Print the dictionary with keys that are words on web pages (w)
Print this menu (m)
Quit (q)

The user can type in the letter for a command, and your program should carry it out. After completing the command, it should continue to request and execute commands until the user quits.

2 Details

Initial input to the program will consist of a “root URL” and the filename of the first web page to be visited. The root URL is an absolute URL for a directory that contains all the web pages the program will visit. In the example above, the root URL would be

    http://cs.usfca.edu/~peter/cs110

and the filename might be

    hello.html

So an absolute URL for the first page visited would be

    http://cs.usfca.edu/~peter/cs110/hello.html

After getting the initial input, the program should use depth-first search to crawl the pages in the directory specified by the root URL. During the crawl it should build the URL–word-list dictionary. Keys should be absolute URLs. After visiting all the files that are reachable by depth-first search, the program should build the reverse-index dictionary.

It should then begin prompting for and executing user commands.

As discussed in class, the HTML pages that will be visited have a very simple format: each line is either an HTML tag or a word in ordinary text. The words will consist only of lower-case letters (no spaces or punctuation). The HTML tags will be

    <html>
    </html>
    <title>
    </title>
    <body>
    </body>
    <a href="...">
    </a>
    <br>

The URLs in the `<a href="...">` tags will be relative to the root URL. So they’ll simply be filenames ending in “html” — e.g. hello.html or goodbye.html.

You can use the Python `requests` module to get the contents of a webpage:

    import requests
    ...
    page = requests.get(url)
Here, url is the absolute URL of the page. The object returned by requests.get has a content attribute. This contains a string with the html. In this example, then,

    page.content

contains the html for the web page.

The search command should prompt for a word to search for. If the word is in the database, it should print a list of the absolute URLs of the pages that contained the word. If the word isn’t in the database, it should just print a message to this effect.

The open command should prompt for an absolute URL, and try to open Firefox at this URL. On the Linux systems in the CS labs, this can be done using the system command from the os module:

    import os

    . . .

    os.system("firefox " + url + " &")

Here, url is the absolute URL of the page. Notes: If Firefox is already running on the system you’re using, you may need to kill it before opening a page from the program. Also, this will probably only work on Linux systems. The ampersand (&) at the end will start the browser in the background; this will allow your program to continue without killing the browser first.

The commands that print the dictionaries should print each key on a single line, and on the lines immediately following the key the elements of the corresponding value (which is a list). For the URL–word-list example, above, the u command should print

    http://cs.usfca.edu/~peter/cs110/hello.html
    hello
    ciao
    goodday

    http://cs.usfca.edu/~peter/cs110/goodbye.html:
    goodbye
    bye
    ciao
    goodday

3 Test Data

I’ll put some test data on the class web site. I’ll include the output of the print commands when a solution to the assignment is run with the test data.

4 Due Date

In order to receive full credit, your program must be in the p4 subdirectory of your Subversion repository by 2:00 pm on Monday, November 21, and you must turn in a print out of your program by 5 pm on the 21st. (Note that this is different from the date in the syllabus.)
5 Grading

Your program will be graded on the basis of its correctness and its “static features.”

1. Correctness will be 60% of your grade. Does your program take input in the correct format? Does it use depth-first search to visit the pages? Are the dictionaries correct? Does it correctly execute the user commands?

2. The following static features will be graded.

   (a) Documentation will be 10% of your grade. Does your header documentation include the author’s name, the purpose of the program, and a description of how to use the program? Are the identifiers meaningful? Are any obscure constructs clearly explained?

   (b) Source format will be 5% of your grade. Is the indentation consistent? Have blank lines been used so that the program is easy to read?

   (c) Quality of solution will be 20% of your grade. Does your solution contain unnecessary calculations? Is your solution too clever — e.g., has the solution been condensed to the point where it’s incomprehensible? Are functions and data structures (e.g., lists, strings, dictionaries) used properly? Are any functions (including the main program) more than 20 lines long?

3. Extra credit. You can get up to 10 points extra credit by augmenting the output of the search command: instead of simply printing a list of URLs, the search command should create an HTML page with nicely formatted links to the pages that contain the word that was searched. After creating the page your the page should be opened in Firefox. An easy way to do this is to copy the page to your CS Department home page, which is stored in

   /home/web/your_userid

   Then, if your page is called results.html, an absolute URL for the page is

   http://cs.usfca.edu/~your_userid/results.html

   and you can open this page in Firefox as described for the open command, above.

6 Collaboration

It is OK for you to discuss solutions to this program with your classmates. However, no collaboration should ever involve looking at one of your classmate’s source programs! It is usually extremely easy to determine that someone has copied a program, even when the individual doing the copying has changed identifiers and comments. If we discover that someone has copied a program, the authors of both programs will receive an F in the course.