Course: TTS Date: 04/10/12 Student ID: 0943941

Summary of Programming and Design Decisions

A successful web crawler should be able to accomplish the following: obey the rules regarding its crawling, detect and maintain a list of visited web sites, utilize a method for handling priority, and function at a high performance. My implementation of this local crawler has fulfilled these requirements, with several key design decisions along the way.

To obey the limitations of robots.txt, I used Python's module 'robotparser'. While it could verify that my crawler could access a link, robotparser does not check either the request rate or crawl delay attributes. To account for these, and make the crawler behave in accordance with the rules, I wrote two regular expressions to determine the values at these points. These times are then included in the crawling function as pauses in execution, or set to zero if not found.

After determining the rules, I fetched the HTML source with urllib. Urllib2 was considered, but urllib had all required functionality, and was smaller. If this web crawler were to be used outside of an academic setting, urllib2 would be preferential for its recentness.

To parse the HTML, I used the find command to search for '<!-- CONTENT -->' and '<!-- /CONTENT -->' within the string, sending the URLs without them to a 'broken URL' list. To do so assumes that the HTML code is well-formed. If the HTML source contains the Content tags, the URL is added to list of visited pages, and the crawl of the page begins.

I used both the software package 'Beautiful Soup' and my own regular expression to find the URLs within the content tags. My regular expression assumes two things: that all href tags will be located within an <a> </a> pair, as per HTML convention, and that a valid URL will be of the form 'http://', 'http://www.' or be a local link. Each valid URL must end in .html, and can only contain alphanumeric characters (& and %, while valid URL symbols, cannot be detected within my regular expression). Both my parser and Beautiful Soup return the same amount of URLs, indicating a comparable performance.

Once a list of successful URLs has been taken from page, you confirm that the link is either of the form http://ir.inf.ed.ac.uk/ or a local link of #.html. This assumes there are no other servers or other forms of the URL that constitute as local. Also assumed is that the end of the URL has numeric values – my code does not contain any checks for otherwise. If the code is not local, the URL is discarded. During tests, I did not encounter any non-local links beginning with href.

For each local URL found, the web crawler checks if it can be parsed based on robots.txt. If it can, it corrects the folder substructures to remove any '../' within. The URL is now valid for adding to the list. The crawler next assigns the priority based on the page number of the URL. Since the frontier queue uses Python's heapq (a decision motivated by speed), popping a URL off the heap will result in the lowest number. Since our task specification involved prioritizing the highest number, the crawler takes one divided by the number – the larger the number, the smaller the priority. This way, the heap can still be used for efficiency, and the priorities can be based on the numbers.

Finally, the crawler checks through both the list of visited URLs and URLs to visit. If the URL does not appear in either list, it is pushed onto the heap with a priority based on the inverse of the page number. This is done last to save computational waste – there is no need to iterate through a list if it can be invalidated in one step.

Other speed concerns have arisen within the list of visited URLs. The crawler currently maintains this as an unordered list for simplicity. If speed were an issue, this list could be updated to lower the search times. However, this is not the bottleneck of the program – the crawl delay is (as showcased in the statistics summary below).

The crawler begins at the seed http://ir.inf.ed.ac.uk/tts/A1/0943941/0943941.html and proceeds to crawl with the following statistics -

Statistics

Here is a brief list of the statistics found during my testing period. Please note:

1) During testing, I ran the code through my home network, to understand the speed limitations of a remote session. Web crawlers would not operate on a local level, so this test confirmed that timeouts/server response time did not affect the code. The results of the session are below as non-local session, with the result slower than the local (as expected).

2) To ensure the correctness of the pages added, I used a function that compared the length of a list (either the list of URLS to visit or already visited) to the length of the set of the list. This method would throw an error if a duplicate link would occur. This function was not present in the statistics below.

3) A comparison between Beautiful Soup and my Parser shows that my parser improves the time by around 30 seconds. I would attribute this to the simplicity of my regular expression (ignoring valid characters and assuming only valid forms of http://). The Beautiful Soup method additionally has another method call between when it receives the trimmed HTML and when it starts its crawl – this could be the incremental result of this call.

4) I attempted other priority metrics to see how they would affect performances. I used the untouched URL number as a priority, so that lower numbers would be dealt with first. This method resulted in far fewer broken links (I believe the reason is because broken links are not added to the already viewed list). The second priority method was based on time - earlier entries receive the number of links the crawler has seen (increases over time). Results are detailed below.

5) Unless otherwise stated, the statistics shown are for both my regular expression and beautiful soup, running on a local machine with normal priority, crawl delay and request rate.

Crawler Runtimes

Local:	919.962 seconds or 1.153 seconds per valid page
Non-Local:	1189.437 seconds
Beautiful Soup:	946.624 seconds
Without Crawl Delay:	12.423 seconds
Without Request Rate:	921.124 seconds
Lower Number Priority:	883.471 seconds (with 64 broken links)
Time As Priority:	848.475 seconds (with 31 broken links)

Alternate priorities had same amount of crawled pages

Page Information

Links Found:	
BeautifulSoup:	18019
MyRegExp:	9910
Outside Links:	0
Pages Discovered:	905 unique pages
Pages Crawled:	798 Distinct Pages
Errors:	38 HTTP Error 505 - Bad Gateways
(standard run)	35 HTTP Error 404 – Not Found (These contained link information for CNN Money, but no content tags, so they were discarded)
	34 HTTP Error 404 – Not Found (These were blank pages without any information)
	107 Total unparsed pages (pages without content tags)

Modules/Software Packages Used - RobotParser, Urllib, Re, Heapq, Time, BeautifulSoup