Yong Liang 23941603
Proj 1
Due: 2/5/2016

(40%) The report discusses all the relevant information.

1. (0% but required) If it is not blatantly obvious, please indicate where in your source code parts A and B exist.

Yes, part A and part B exist as links100.txt and links1000.txt

2. (15%) Description of system, design tradeoffs, etc.
   - Focus on how you implemented the high level pieces discussed in the textbook and class, rather than on how a crawler works in general.

For the data store, I used a hashmap to keep track of what links I have already seen. Using a simple hashmap can quickly keep track of what links has been visited. Alternatively, a sql database would be able to store much more information, but requires lot more schema setup.

Then I use a queue to store the links for processing, and a counter to exit the program when the desire number of page crawled is reached.

I also have variables to keep track of number of documents scanned, set of robots sites, set of sitemap sites, and a list of restriction link resource paths.

The main structure of the program starts with the crawl() function, which dequeue the links one by one and wait 5 seconds per crawl.

For Parser() function, it check&grabs the current site for robots.txt and sitemap. Then make a connection to the site and start parsinglink() on the body for outgoing links.

For ParserLink(), it looks into all the extracted links and calls the validation function to make sure they are valid links of html or pdf.

For Validate_add_url(), it first stems url to remove ? or # parameter links. Then it checks if its html/pdf, check if its already_seen(), and check if it respects_robots().

Already_seen() checks the url variations http/https/www with the hashmap of unique links already seen by the program.

respects_robots() consist of all the baseurl+resource path that was gather from robots.txt. All the links would check against this file to make sure they respect disallow paths.

3. (5%) List the software libraries you used, and for what purpose.

- Implementation must be your own work, (no crawling libraries) but it is OK to use helper libraries (e.g. regex, HTML parsing, or HTTP). If you are unsure, please ask.

```
import java.io.*;
import java.util.*;
import org.jsoup.*;
```

I used IO to output the resulting links into a txt file.

I used util data structures link Hashmap, ArrayLIst, and Queues.

I used Jsoup to make HTTP connections and get requests for webpage header information and body text. This also useful to selection links from the DOM struction in HTML file.

(5%) Discuss how you parsed robots.txt files.

- How many did you find?
- In part B, did you notice any problems with robots.txt on other domains?

I found 2 robots.txt for both part A and 6 for part B.

In Part B, robots.txt contains many article links for new stories, some of the problem I had was links that are broken.

Some of the sites have both disallow and allow in USER_AGENT, might be overlapping, and I'm not sure how to map the exact relationship in this crawler, so I just ignore the Allows to be safe.

```
crawled docs: 6
links on the queue: 96/101
num of robots found:2
num of sitemap found:2

crawled docs: 22
links on the queue: 980/1001
num of robots found:6
num of sitemap found:2
```

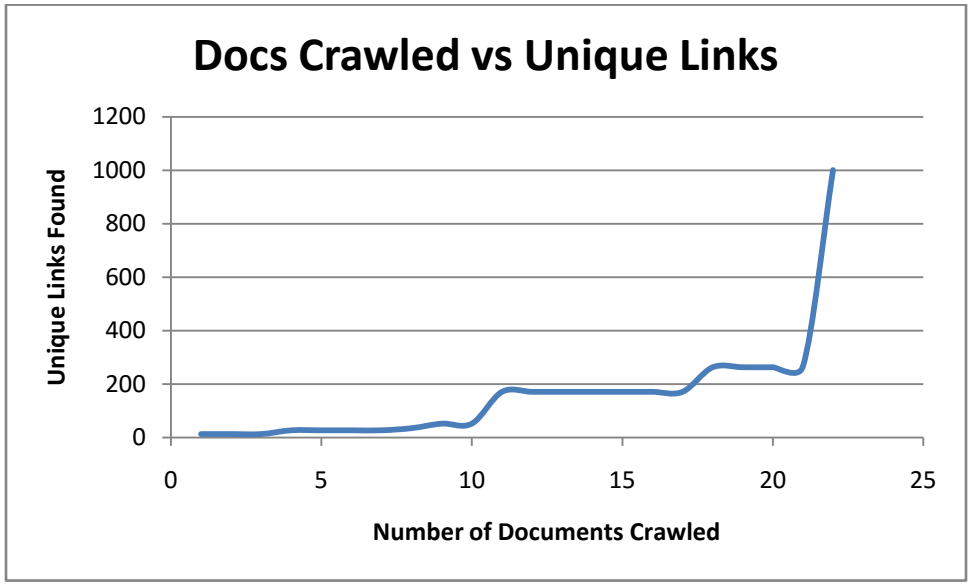
4. (5%) There are certain kinds of websites that end up being self-loops. They have links to themselves with different parameters, or with a slightly different URLs, but have an endless number of web pages to return. One example of these pages are calendar pages. Give another example of a kind of page that might break your assumptions. How might you deal with these pages in a more sophisticated crawler?

For many of those website they have a pattern where the parameter of the dates are external with ?date=xxx or #january. These I use link stemming to remove the extra text.

Some of the more difficult are with number built into the links http://www.date.com/2018/1/1/. These might be built with quick relational scaffolding with framework like Ruby on Rails. I would use the SIMHASH algorithm discuss in class to detect duplication/ similar documents.

5. (10%) Plot a graph of Unique Links vs. Number of Documents Mined (***Part B***)
   - Y Axis from 0 to ~1000 Links
   - X Axis from 0 to ?? Documents
   - Explain the shape of this graph
     - Why does it look this way?

```
crawled docs: 22
links on the queue: 980/1001
num of robots found:6
num of sitemap found:2
```



The graph based on my crawler shows many period of plateau, then at the end hit a jackpot of outgoing links. From my research, its seems website within the same domain basically already explored most of the links, until we reach a new external link, we won't see much increases in the unique links. This is also caused by having read all the links in the SITEMAP. Therefore, we have a period of crawling through pile of these internal links.

The SITEMAP is also explaining the ending spike of unique links. The crawler hit a new sitemap with ton of articles, therefore all valid unique links.