GEORGIA INSTITUTE OF TECHNOLOGY, COLLEGE OF COMPUTING

CS 6675: Programming Assignment 1

Abhishek Sen, GTID: 902898709

January 30, 2016

WRITE A WEB CRAWLER OF YOUR OWN

DESIGN OF THE CRAWLER

MOTIVATION

I love technology and I love to read. 'The Google Story' by David A Vise is a great book that recounts Google's journey through the years, its inception and rapid rise. I had enjoyed reading the book a lot and was keen on finding similar books to read. I wrote a simple web crawler of my own using Python to do just that.

DESIGN

Goodreads is a social cataloging website for books with the tagline 'Meet your next favorite book'. The website has a very comprehensive catalog of books where readers can rate and review books. An interesting feature of the website is the 'Readers also enjoyed' section which I think is an attempt at recommending similar books using collaborative filtering. However, the section shows only five to ten similar books at the most on a page. I would rather have a huge list of books similar to my favorite book. My spider accepts a seed url, gets the source HTML and parses that to grab the 'Readers also enjoyed' section. I achieve this using BeautifulSoup which is a very handy Python HTML parsing library. Then the spider goes on to perform an aggressive Breadth First Search(BFS) by selecting three similar books from the 'Readers also enjoyed' section and expands the Crawl-Graph in realtime. I call it 'aggressive' because it selects only a limited number of similar book pages to crawl, (parameterized as 'branching-factor', I choose 3 in my code), as opposed to a 'crawl all similar book approach' as a regular BFS might have done. It stores the link structures in a text file in a 'BookA ->

BookB' format which implies 'BookA' led to 'BookB' in the web crawl. This allows me to retrieve all the book names from the file once the spider has finished crawling. Never again will I have to be confused about what book to read next, or so I wish.

VISUALIZATION

I create a visualization of the web-graph crawled by the spider using D3, a visualization Javascript library. This allows me to visualize and understand similarities in the book graph. Every Breadth first layer (nodes connected to the same node) are similar in themes. The themes vary as we go from one layer to another.

PROS

The aggressive BFS discovers books varying in subject matter yet it stays true to themes like Technology, Silicon Valley, Software, Entrepreneurship, Internet Computing and similar subjects. The idea is very simple but solves the problem I intended it to solve.

On going through the crawl graph, I found that uncovered books not only like 'The Second Coming of Steve Jobs' and 'The Wikipedia Revolution: How a Bunch of Nobodies Created the World's Greatest Encyclopedia' that are very related to our origin book subject matter, but also books like 'The Human Brain: A Guided Tour' and 'An Imaginary Tale: The Story of the Square Root of Minus One', that are not very related yet are relevant to the central themes. This brings variety to my reading list. The branching-factor can be controlled to control the variety in the subject matter. The higher the branching-factor, the more similar the books are to each other, while, the lower the branching-factor the higher the variety in the themes of the books.

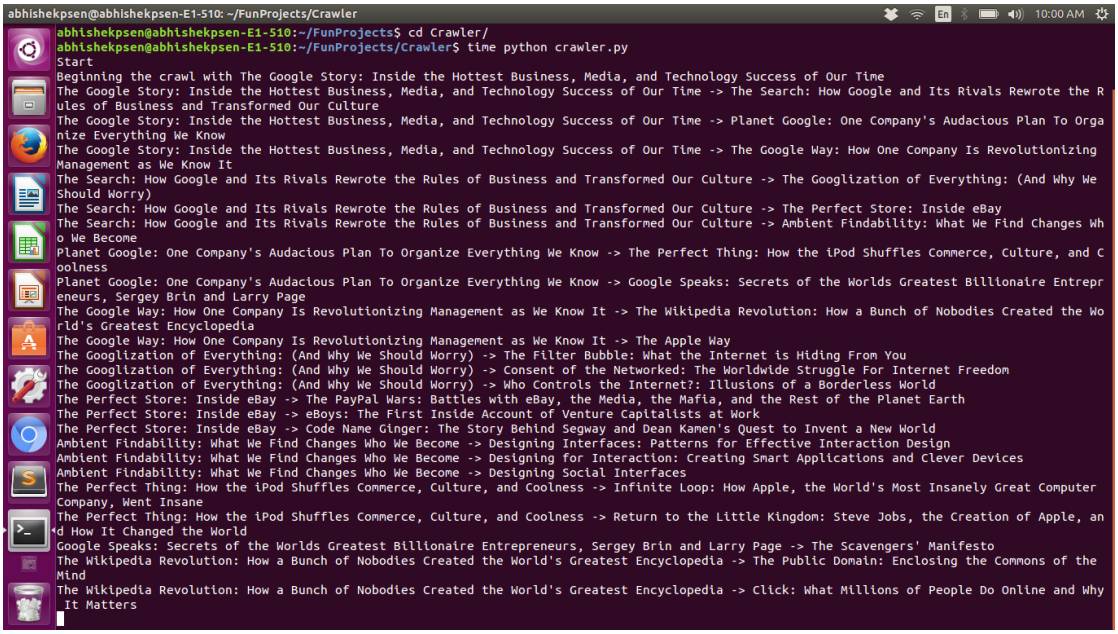
I also investigated using a Depth First style Crawler, the pages crawled in that case was, as one would expect had a much greater variety in subjects since DFS is a more aggressive search than BFS.

CONS AND POSSIBLE FIXES

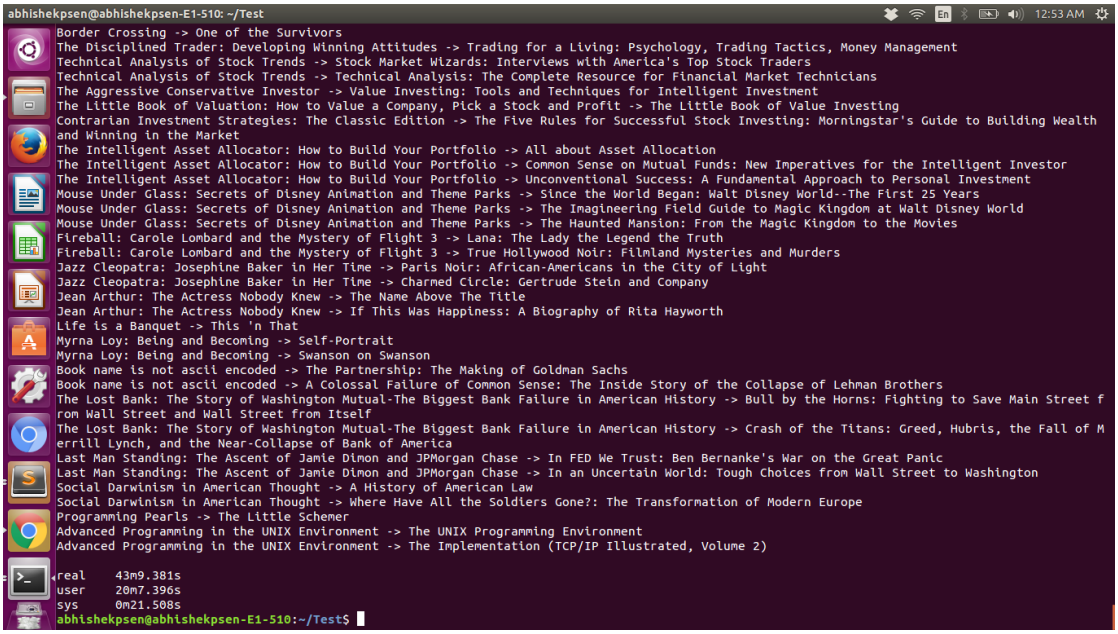
The spider is slow. It takes about thirty minutes to crawl 1100 pages. The statistics are presented in the crawl statistics section. A caching technique could be used to speed it up.

The spider discovers some unrelated books at times. The farther we go from the origin (seed) book, the more the difference. For instance, 'Manliness and Civilization: A Cultural History of Gender and Race in the United States, 1880-1917' is not really a book I would believe is similar to 'The Google Story', but was uncovered by the spider. However this can be fixed by fine tuning the branching factor.

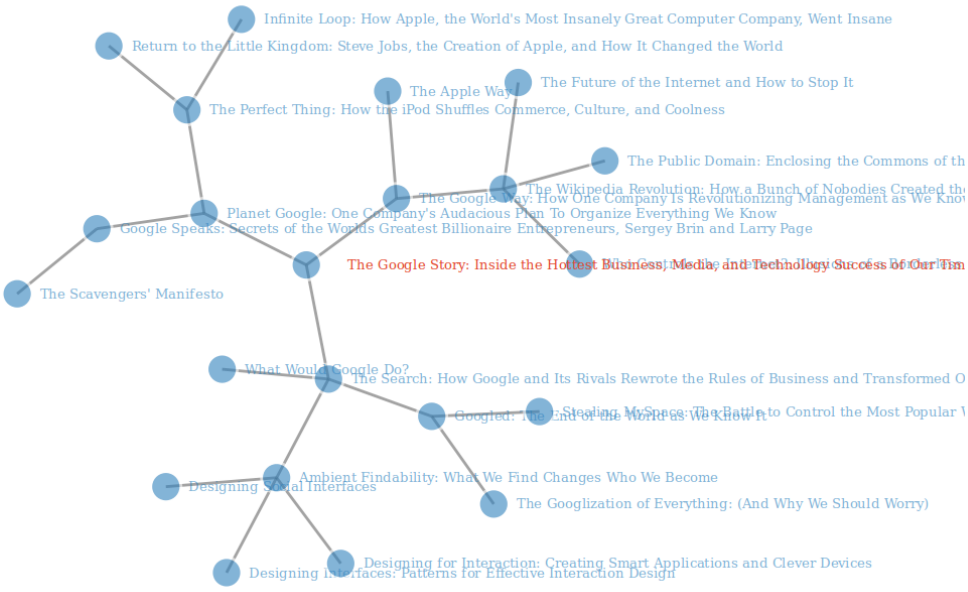
SOME SCREENSHOTS



The crawl begins with our seed book url and then proceeds to find similar books



It took about 43 minutes to crawl 1500 pages

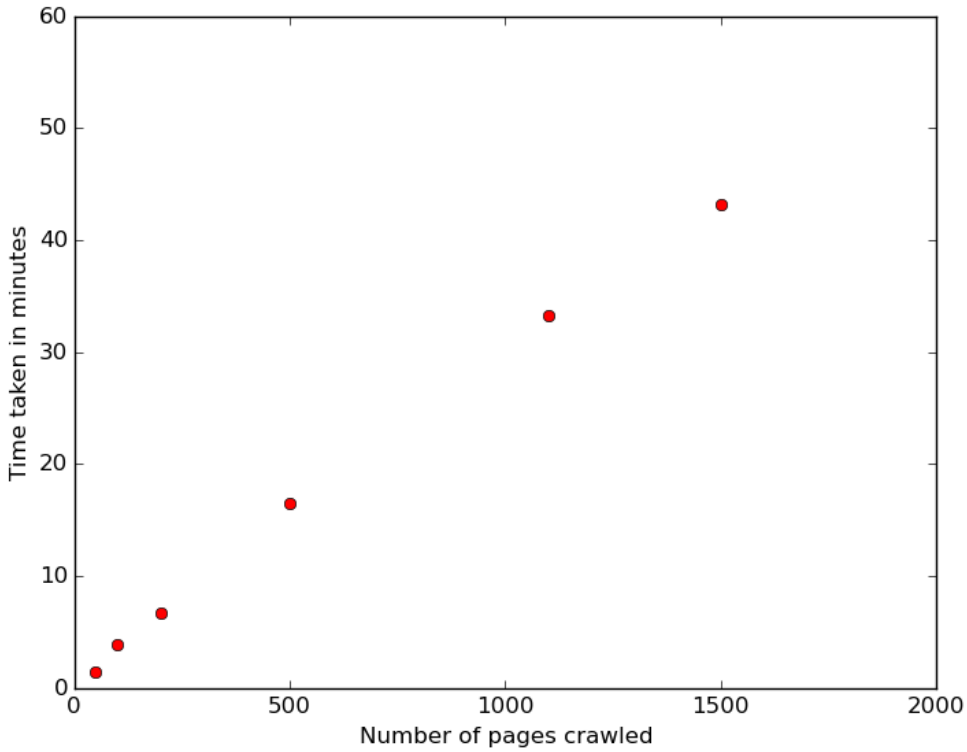


Visualizing the crawl graph using D3

Crawl Statistics

The spider crawls at the rate of 35 pages a minute on an average.

Number of pages crawled	Time Taken(minutes)
50	1.5
100	3.9
200	6.67
500	16.5
1100	33.3
1500	43.16



Time taken to crawl as a scatter plot

Precision for a focused crawler is the percentage of the relevant web-pages it has crawled out of the total pages crawled. Recall is what percentage of relevant web-pages were actually crawled. The precision and recall for this particular focused crawler is very subjective because currently I don't have a quantitative measure for measuring the relevance of a web-page. Two books that seem seemingly unrelated from the title might actually be related. One would have to read them to understand.

EXPERIENCES AND LESSONS LEARNT

I found it particularly interesting to write a focused crawler because I used it to solve a problem of my interest. The crawler would crash often, particularly on calling requests.get method. I could make it more robust using exception handling. I applied a modified version of BFS to add variety to my crawled pages set. I also learnt to use BeautifulSoup which is a very useful HTML scraping tool.