

Project Report

April 28th, 2016

By: Luis Liang

Nathan Lin

Mauricio Rivera

Anahit Sarao

Web Crawling

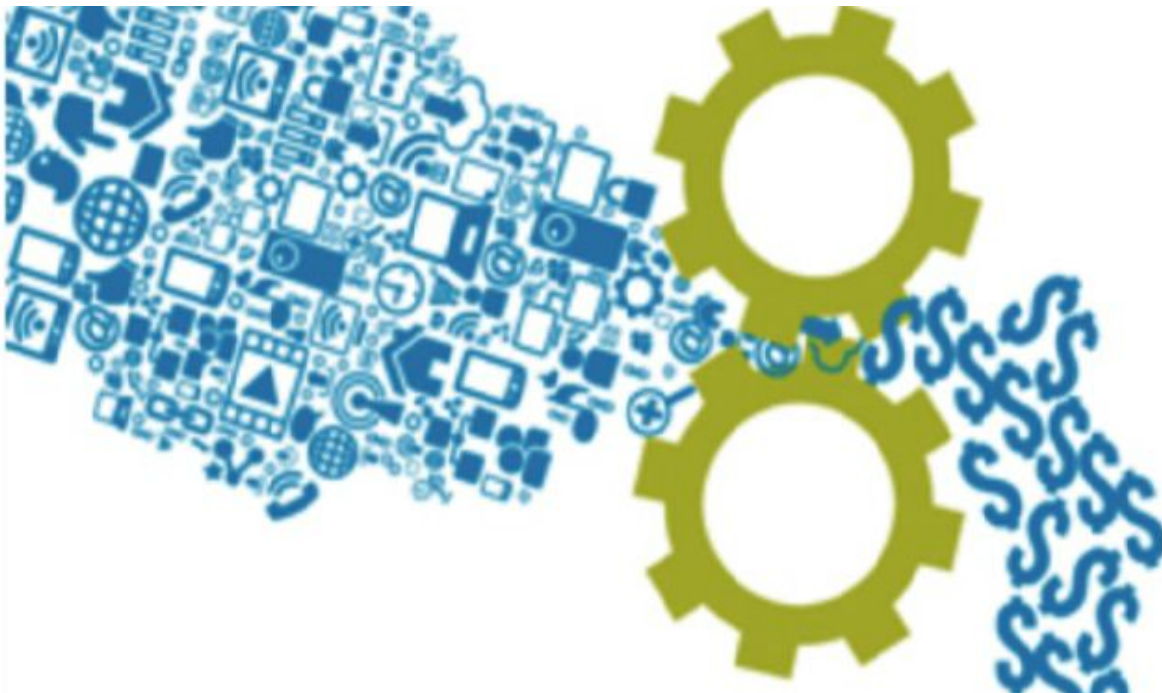


Image retrieved from: <www.connotate.com/wp-content/uploads/BigData_Gear-672x372.png>

Table of Contents

- Abstract
- Introduction
- What is a Web-Crawler?
 - What are Web-Crawler used for?
 - Typical Web-Crawlers
 - How can Web-Crawlers be improved?
 - Google Web-Crawler analysis
- Web-Crawler Design & Analysis
 - Testing
 - Difficulties
 - Improvements
- Conclusion
- References
- Appendix
 - Web-Crawler source code

Abstract

As the internet became part of our daily life, more and more web-browser are emerging the web, where we all continuously search the web for answers, entertainment, etc. Due to this exponential increase, the data must be structurize to ease the search to accommodate the majority of the users. In other words, web-crawling is an important tool (data structure algorithm) on today's world-wide-web, as more and more sites becomes available, web-crawling performance must improve, too. Otherwise, this structure will lose its purpose.

On this project, we will focus on the complexity and variety of web crawling algorithms. There are different applications that may be used to crawl the web, for example: search engines, web analytics (SaaS), and even for cyber security.

The report will include a detailed report about different ways we can implement a web crawling algorithm, its complexity and future applications that may be help internet users. There will also be a basic web crawling demo that will illustrate how 'spidering' works around the web.

Introduction

Web crawling is a common algorithm that people use every day. The most popular one would be the Google Search Engine. In general, people can type some key-words in the Google search bar, and when hit the “Enter” key on the keyboard, hundreds of web links will pop out. One thing that can make this happen is called web crawling algorithm.

In the following report, we will discuss the web crawling algorithm in detail. It will include the theoretical part and the practical part. For theoretical part, the functionality of the web crawling algorithm will be explained, and the searching on how to creating an effective web crawling algorithm will also be introduced. In practical part, a pseudocode and actual code will be create and test on the computer. The testing results will be analyzed as well. In addition, we will also focus on how to improve the code to be better after researching a proper solution online.

What is a Web Crawler?

A web crawler is also called a web spider. When most people talk about internet search engines, they really mean World Wide Web search engines. Before the Web because the most visible part of the internet, there were already search engines in place to help people find information on the Net. Programs with names like “gopher” and “Archie” kept indexers of files stored on servers connected to the internet (Franklin).

In general, before a search engine can tell users where the files or documents is, it must be found. Otherwise, users are going to wait while the search engine is searching the possible result. In order to find useful information among millions of web pages, a search engine must use some special software robots to help it store those web pages on a list. And that software is called spiders. And when a spider is building lists which contains millions of web pages, the process is called web crawling (Franklin).

How does a Web-Crawler works?

Before exploring how a web crawler work does specifically, it is necessary to talk about a search engine because a web crawl is a software that belongs to a search engine. The role of search engines on the internet is to select which piece of the internet web page should be used, based on a few key words. They keep an index of the words they find, and where they find them. A successful search engine must have a large list of millions web pages and also need to keep them updated in time.

Let us focus on the web crawler now. The process of how a web spider start its travel is to start with the heavily used servers, which means the web spider will go over the most popular web pages and see if these pages contain the key word or combination of keywords. If they do, the web spider will continue indexing the works on these pages and following every link that can be found within the site. In this way, the crawling system will quickly travel and spread out across the most widely used portions of the web (Franklin). Web crawlers can copy all the pages they

visit for later processing by a search engine which indexes the download pages so the users can search much more effectively.

Imagine that web is a graph, and pages can be represented by nodes, and links between that page and another page can be considered as one or many edges. In this case, a spider performs a graph traversal, and this can be done in many ways. However, the central piece of the web crawler is a priority queue. It also called frontier, which stores the lists of the pages (URLs) what it needs to crawl next in some order. The sequence of the page list will be discussed later on. The way that a typical web crawl will look like is to take the first item out of the priority queue inside the loop, whichever has the highest priority at the moment. Then, the crawler will fetch the page, which means it will get the string which represents the html of that page, and it saves the string somewhere for processing. However, the crawler should not try to change the data. Instead, it should just index the data and states that the crawler has found the data. Next, the crawler will go over each html string and extract all the hyperlinks which means all the URLs that can be recognized. Since getting all the URLs, the next step is to push them on the priority queue. Therefore, the priority queue is the one who will determine the order of the URL list. In general, there will be another step between extracting and pushing because it does not need to push a link that has been already processing. In other words, it should not be duplicated links on the priority queue. When the processing has been finished, and that is just one iteration of the loop. Then the web crawler will pop the next entry of the priority queue and keep going like this. The can be reflected to different things mainly depends on the different tasks. However, it will mostly reflect on the ‘age’ of the page, which means how long has it been crawled. The other to say it more accurately is what the chance of that page is going to be changed since the last it has been crawled. It can be throwing another question out. How to estimate the number.

Typical Web-Crawlers

Today’s crawlers use a lot of machine learning work to search the web, it usually involves a site-ranking system by their relevance and the amount of times people use that site. This plays an important role in search engines because most of the times, users go for the “top” resulting sites.

However, the web is growing and changing rapidly, so search engines cannot keep up with the ranking-system they use.

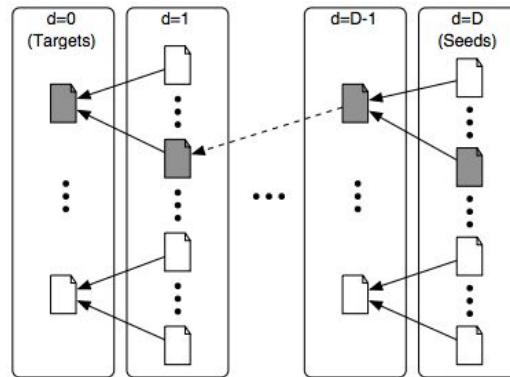


Figure 1: Basic crawler with seed page and targets(gray)

The most commonly used crawlers (or adaptations of them) are Breadth-First, Best-First, PageRank, Shark-Search, and InfoSharks. These frameworks have been shaping the way people or bots crawl the web.

Breadth-First is the most simple strategy to crawl, being one of the first crawlers ever to hit the web. It works with a FIFO queue, crawling through links as they come. This crawler does not use any type of topics or keywords.

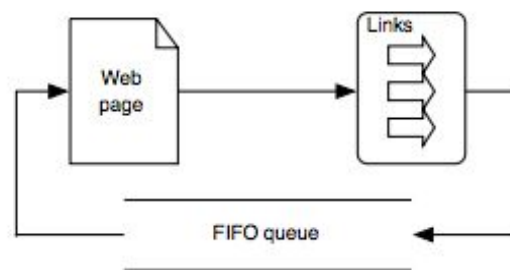


Figure 2: Breadth-First Crawler

Page Rank was first introduced in 1998 and the crawler worked based on the sites the user viewed. A page's score, or rank, is based on different sites that point back to the site. The

PageRank is used by Google search engine, since the rank reflects a site's quality and a good probability that the user will come back to top ranked sites.

Google Web-Crawler analysis

Google Web-Crawler, also known as Googlebot, it is imagined as a little spider crawling the web for sites. However, in reality, Googlebot sends request to the web browser, and then it is handed to the indexer for processing. To avoid overwhelming the servers, Googlebot request much less than its capability, although it already request and fetches thousands of sites at once already. From the indexer, the site is downloaded alphabetically from search term. To improve the speed performance, the indexer also ignores common words, such as the, is, etc (googleguide site).

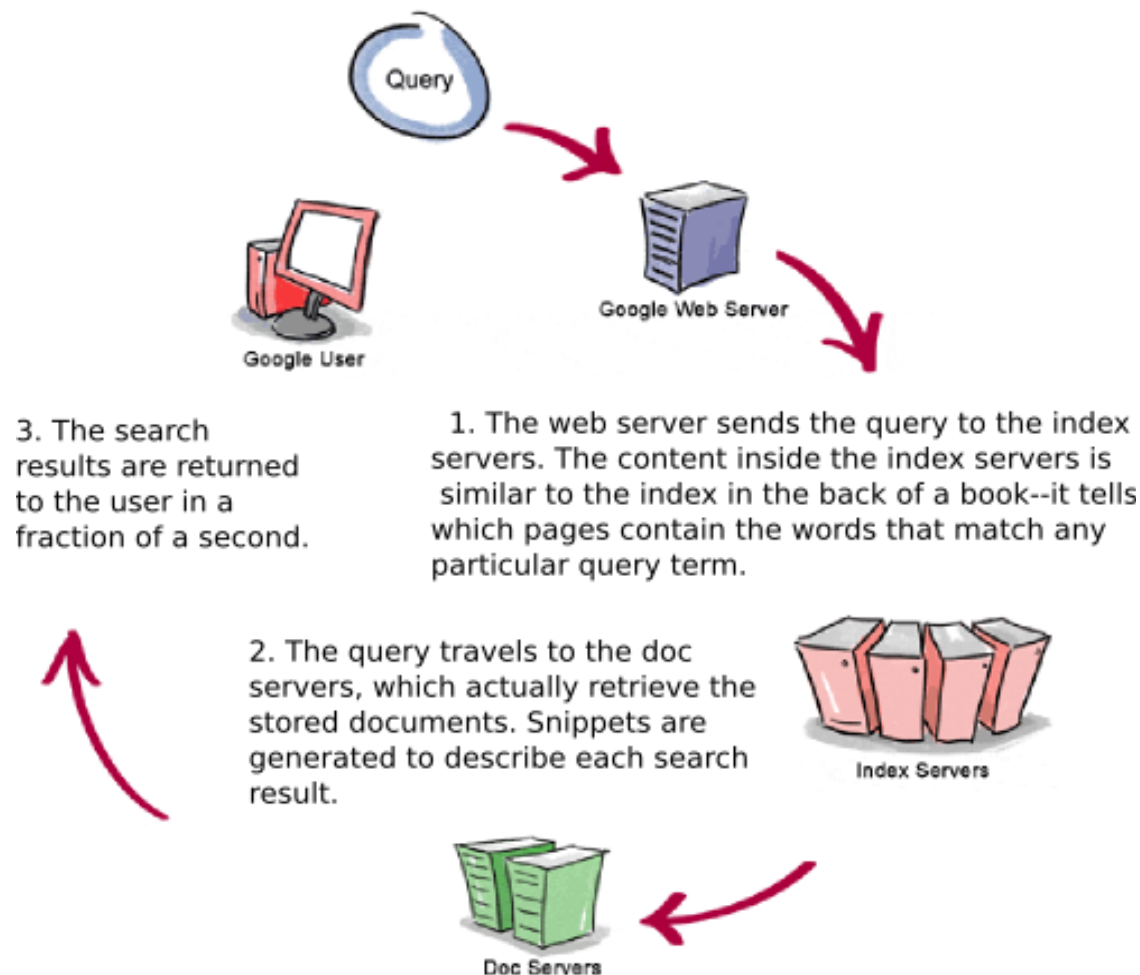


Figure 3. Google Web-Crawler Graph

Image retrieved from <Copyright © 2003 Google Inc.>

Web-Crawler Design & Analysis

The web crawler was written in Python code. The code used HTML Parser and URL API from the Python library. The web crawler simply connects to the URL provided and searches for the word within the site. There is a failsafe for the URL connection and for failed word search. If the word is not found it will not be cached and the URL will be blacklisted as already searched. If the word is found the URL is whitelisting and returned on the console.

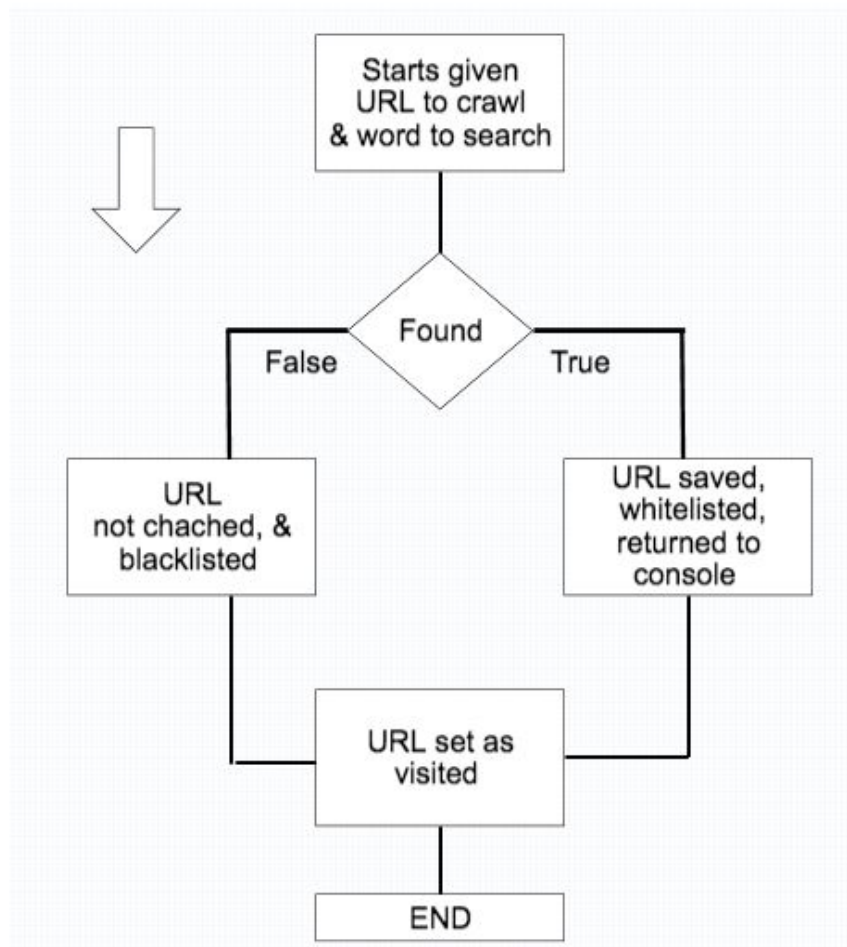
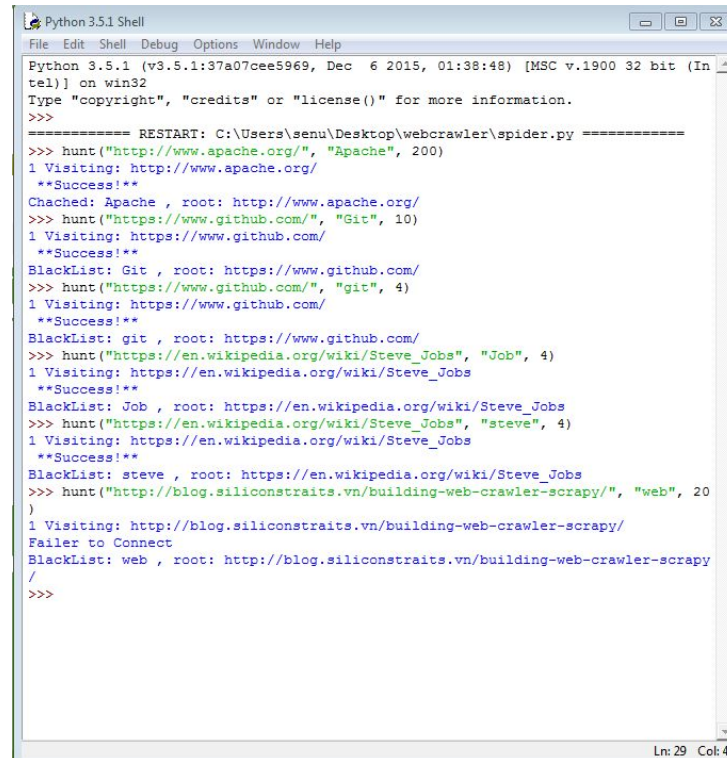


Figure 4: Flowchart of Web Crawler Algorithm

Testing

Python build 3.5.1 was used to create this web crawler. The testing environment is [GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin. To load the script within the Python interpreter, the command `exec(open('./spider.py').read())` was used.



```
Python 3.5.1 Shell
File Edit Shell Debug Options Window Help
Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:38:48) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\senu\Desktop\webcrawler\spider.py =====
>>> hunt("http://www.apache.org/", "Apache", 200)
1 Visiting: http://www.apache.org/
**Success!**
Chached: Apache , root: http://www.apache.org/
>>> hunt("https://www.github.com/", "Git", 10)
1 Visiting: https://www.github.com/
**Success!**
BlackList: Git , root: https://www.github.com/
>>> hunt("https://www.github.com/", "git", 4)
1 Visiting: https://www.github.com/
**Success!**
BlackList: git , root: https://www.github.com/
>>> hunt("https://en.wikipedia.org/wiki/Steve_Jobs", "Job", 4)
1 Visiting: https://en.wikipedia.org/wiki/Steve_Jobs
**Success!**
BlackList: Job , root: https://en.wikipedia.org/wiki/Steve_Jobs
>>> hunt("https://en.wikipedia.org/wiki/Steve_Jobs", "steve", 4)
1 Visiting: https://en.wikipedia.org/wiki/Steve_Jobs
**Success!**
BlackList: steve , root: https://en.wikipedia.org/wiki/Steve_Jobs
>>> hunt("http://blog.siliconstrait.com/building-web-crawler-scrapy/", "web", 20)
1 Visiting: http://blog.siliconstrait.com/building-web-crawler-scrapy/
Failer to Connect
BlackList: web , root: http://blog.siliconstrait.com/building-web-crawler-scrapy/
>>>
```

```
===== RESTART: C:\Users\senu\Desktop\webcrawler\spider.py =====
>>> hunt("https://www.github.com/", "git", 1)
1 Visiting: https://www.github.com/
**Success!**
BlackList: git , root: https://www.github.com/
>>> hunt("http://www.apache.org/", "Apache", 200)
1 Visiting: http://www.apache.org/
**Success!**
Chached: Apache , root: http://www.apache.org/
>>> |
```

This screenshots of the Python shell as it visits a web page the algorithm looks for the key word and will either return a blacklist or whitelist URL.

Difficulties

The hardest part of designing this particular crawler was learning the different python libraries (modules) we had to use. Python has many libraries to use, in particular, we needed a module that would help us connect to the web and access different URLs. Also, our group is not too familiar with Python, so it was a learning process for all of us. Besides that, there were sites that helped us understand the basic of idea of how to code a basic web crawler from scratch. These difficulties motivated us to do in-depth research and code as we learned more.

Improvements

To have a more complex crawler, if we had the chance to, we would research more algorithms to follow. This way we can present an web crawler that is more complex. As far as the entire report, the group had to be flexible with the scheduled plan done during the proposal because member had different things going on. This set us behind schedule for a bit, but in the end, we all participated and contributed to the team. Other than that, and based on our lack of experience on the topic, we were able to create a working crawler and a detailed report.

Conclusion

To finalize, through our research, we learned the importance that a web crawler has in all aspect of the fast-growing web. A crawler can be used to improve cyber security, search engines and even e-commerce. We visited some general crawlers that are often used like the PageRank and Breadth-First. Also, a detailed explanation and analysis of the Google crawler was given based on the research done online. Then, after extensive research and trial-error practice, we were able to build a basic web crawler that looked through a site and determined if a given word was present. Overall, web crawlers keep getting more complex and are now being flexible, adjusting to how users browse the internet to keep relevant/trustworthy sites on top. As the internet grows, the web crawler will adapt and keep inspecting sites that'll help the users consume information.

References

- Castillo, Carlos, Mauricio Marin, Andrea Rodriguez, and Ricardo Baeza-Yates. "Focused Web Crawling." *SpringerReference* (n.d.): n. pag. *Scheduling Algorithms for Web Crawling*. Web. 24 Apr. 2016.
- "Crawling & Indexing – Inside Search – Google." *Crawling & Indexing – Inside Search – Google*. N.p., n.d. Web. 26 Apr. 2016.
- Frankin, Curt. "How Internet Search Engines Work." *HowStuffWorks*. InfoSpace LLC, 27 Jan. 2000. Web. 27 Apr. 2016.
- "How Google Works - Google Guide." *How Google Works - Google Guide*. N.p., n.d. Web. 27 Apr. 2016.
- "Implementing an Effective Web Crawler." *Implementing an Effective Web Crawler*. N.p., n.d. Web. 27 Apr. 2016.
- Janbandhun, Rashmi, Prashant Dahiwal, Issn:, and 488. "Analysis of Web Crawling Algorithms." *International Journal on Recent and Innovation Trends in Computing and Communication* 2.3 (2014): 488-92. Web. 26 Apr. 2016.
- Kumar, Rahul, Anurag Jain, and Chetan Agrawal. "SURVEY OF WEB CRAWLING ALGORITHMS." *SURVEY OF WEB CRAWLING ALGORITHMS* (n.d.): n. pag. Web. 24 Apr. 2016.
- Menczer, Filippo, Gautam Pant, and Padmini Srinivasan. *Topical Web Crawlers: Evaluating Adaptive Algorithms* V (2004): 1-38. Web. 26 Apr. 2016.
- Shah, Shalin. "Implementing an Effective Web Crawler." *Implementing an Effective Web Crawler*. N.p., n.d. Web. 22 Apr. 2016.

Appendix

Web-Crawler source code:

```
from html.parser import HTMLParser
from urllib.request import urlopen
from urllib import parse

class LinkParser(HTMLParser):

    def handle_starttag(self, tag, attrs):
        if tag == 'a':
            for (key, value) in attrs:
                if key == 'href':
                    newUrl = parse.urljoin(self.baseUrl, value)
                    self.links = self.links + [newUrl]

    def getLinks(self, url):
        self.links = []
        self.baseUrl = url
        response = urlopen(url)
        if response.getheader('Content-Type')=='text/html':
            htmlBytes = response.read()
            htmlString = htmlBytes.decode("utf-8")
            self.feed(htmlString)
            return htmlString, self.links
        else:
            return "",[]

def hunt(url, word, maxPages):
    Visit_Pages = [url]
    Visit_Number = 0
    KeyWord = False
    while Visit_Number < maxPages and Visit_Pages != [] and not KeyWord:
        Visit_Number = Visit_Number + 1
        url = Visit_Pages[0]
        Visit_Pages = Visit_Pages[1:]
        try:
```

```

print(Visit_Number, "Visiting:", url)
parser = LinkParser()
data, links = parser.getLinks(url)
if data.find(word)>-1:
    KeyWord = True
    Visit_Pages = Visit_Pages + links
    print(" **Success!**")
except:
    print("Failer to Connect")
if KeyWord:
    print("Chached:", word, " root:", url)
else:
    print("BlackList:", word, " root:", url)

#Link Into local Int-python
#Python 3.5.1 (default, Jan 22 2016, 17:08:33)
#[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
#exec(open('./webC.py').read())

```