

A project report on

MULTI-THREADED WEB CRAWLER

Submitted in partial fulfilment for the award of the degree of

B.Tech. in Computer Science and Engineering

by

AJITESH SARANATH K 19BCE1558

KOUSHIK SRIRAM 19BCE1629

G O NARENDRA 19BCE1082



VIT[®]

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
(SCOPE)**

June, 2021

ABSTRACT

With the continuing growth of network information technology, there is a vast volume of unstructured data on the network known as *big data*. Because human resources for information collection are time-consuming, web crawler technology was developed. Crawlers are primarily comprised of downloads, information extractors, schedulers and crawl queues. Scheduler will seed the URL provided for download, then downloader will get page information from the Internet to find and send information extractor, extractor strategy according to the instruction from information extraction to obtain information and the next level in the URL, then the next level URL to a waiting queue, waiting queue to go to submit the URL of the heavy, filtering and sorting operation in the URL.

Characteristics of a crawler are given below:

Distributed: Can be synchronized across multiple machines in a distributed environment.

Scalability: Crawling is slow due to the large amount of data, which can be improved by adding additional machines or increasing network bandwidth, performance and efficiency – a web crawler that traverses a site for the first time can download all available files, enabling system, resource utilization efficiency i.e., processor, storage and network bandwidth.

Quality: Web crawlers should give priority to obtaining high-quality pages that users need and improve the accuracy of obtaining pages and reduce the acquisition of other pages

Freshness: Keep search engines fresh, crawling their data independently based on the change frequency of each page and database and crawling new URLs that are randomly created or updated. For example, news, updated novels.

Extensibility: Designed to be extensible crawlers, with crawlers set up in a modular architecture, so as to adapt to the new data format and acquisition protocol.

TABLE OF CONTENTS

1. Objective
2. Proposed System
3. Advantages of Proposed System
4. System Architecture
5. DFD Diagram
6. Conclusion
7. References
8. Appendix

1.OBJECTIVE

Most web crawlers do not and are not intended to crawl the entire publicly available Internet; instead, they choose which pages to crawl first based on the number of other pages that link to that page, the number of visitors that page receives, and other factors that indicate the page's likelihood of containing important information.

The Internet is always evolving and growing. Because it is impossible to know the whole number of web pages on the Internet, web crawler bots begin with a seed or a list of known URLs. They begin by crawling the web pages at those URLs. As they crawl the websites, they will discover linkages to other URLs, which they will add to the list of pages to crawl next. Given the large number of online pages that may be indexed for search, this process might carry on indefinitely.

Network data has grown significantly since the advent of network technology. The Internet has a plethora of large data, and the Internet itself is a collection of these massive data. This data, however, is difficult to keep in a local database for access and processing. People's daily search tools, such as Google, can only offer approximate search results and cannot deliver correct information. To compensate for the shortcomings of general search engines, a search tool that can gather information in a specific direction- a vertical search engine- has arisen. Web crawlers play a vital role in network data collection. A web crawler is a computer application that crawls and indexes hyperlinks. How to make crawlers more accurate and faster to acquire information has become an important study path in the field of crawlers, attracting wide attention from many academics both at home and abroad.

We have developed a multi-threaded web crawler which would crawl two premier websites of VIT i.e., **VTOP** and **CodeTantra**.

2.PROPOSED SYSTEM

In our approach, for every website we crawl, a separate directory/folder is created. Furtherly, every page will be queue and will be crawled one by one. Links will be added and removed from the crawled pages, as and when required. Due to file manipulation, the crawler in Python becomes slow. To boost the crawler, we convert the contents of the file to a set and vice-versa. Once, this type of cleansing the data into uniformity is done, HTML pages are parsed and links are parsed. We use a **Spider** which crawls the entire website and saves the links in the website as crawled and queued. The crawled one's are discarded as soon as possible and the queue one's are crawled the next time the spider crawls the internet, here the VTOP and CodeTantra.

3.ADVANTAGES OF PROPOSED SYSTEM

The suggested approach makes no use of a database. As a result, there will be no need for a duplicate check, resulting in faster data processing. Because of the human approaches employed to increase the crawler, the web crawler is quick. The web crawler makes advantage of several threads. It will prevent links from being traversed in a cyclic fashion. To avoid tampering with the data we have, we keep them in different folders. Our crawler will be substantially quicker since, as mentioned below, we employ a **focused web crawler** and eliminate practically all link traversal.

Focused Web Crawler: A focus web crawler is also called a topic web crawler, unlike general crawlers, focused crawlers only crawl specific web pages, which can save a lot of time, disk space and network resources. As the saved pages are fewer and better, the update speed is faster, which is more suitable for some enterprises and individuals to collect some specific information. The difference between focus crawler and general crawler lies in two modules to filter web links: web page decision module and URL link priority ranking module to filter web pages.

- **Web Page Judgement Module:**

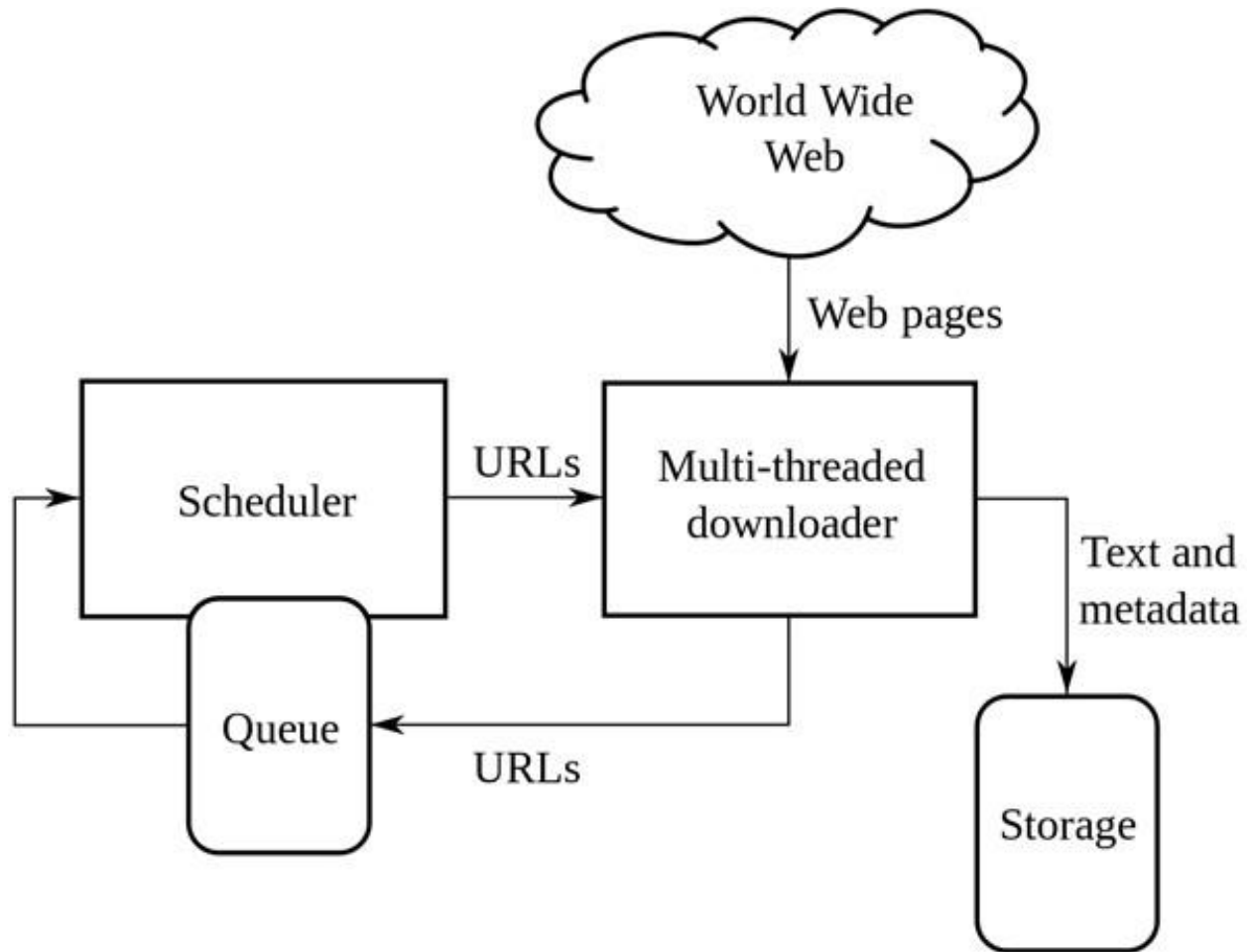
When the crawler crawls to obtain specific content, the web page relevance evaluator starts to compare the relevance between the content in the web page

and the pre-given topic. If the relevance of the web page does not match the previous set threshold, it will abandon the web page to main the high accuracy of obtaining the web page.

- **URL Link Priority Ranking Module:**

This module is mainly used to compare the degree of relevance between the URL resolved and a given topic. The module prioritized the links according to the authority of the links to the content and the number of citations of the links. Sort by priority and remove links that are too low in priority.

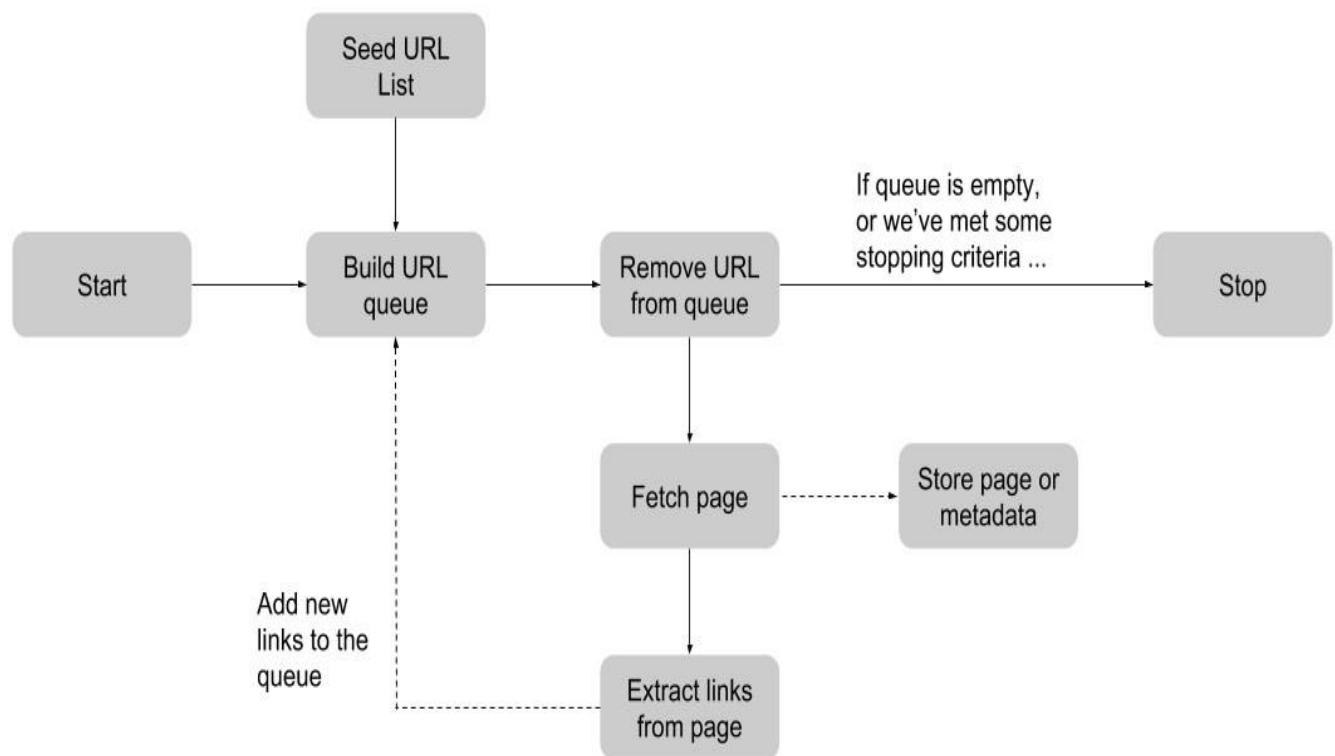
4.SYSTEM ARCHITECTURE



A web crawler is a program that, given one or more seed URLs, downloads the web pages associated with these URLs, extracts any hyperlinks contained in them, and recursively continues to download the web pages identified by these hyperlinks. Web crawlers are an important component of web search engines, where they are used to collect the corpus of web pages indexed by the search engine. Moreover, they are used in many other applications that process large number of web pages, such as web data mining, comparison shopping engines and so on. Despite their conceptual simplicity, implementing high-performance web crawlers poses major engineering challenges due to the scale of the web. In order to crawl a substantial fraction of the surface web in a reasonable amount of time, web crawlers must

download thousands of pages per second and are typically distributed over tens or hundreds of computers.

5.DFD DIAGRAM



6.CONCLUSION

So far, researchers have done a lot of research on the theme web crawler, but there is still a lot of room for research on the performance of theme crawler, web crawlers are all fixed search strategies. Faced with the different forms of web page organization among different websites in the Internet, fixed search modes cannot be effectively crawled. How to improve the performance of them crawlers by integrating crawling rules remains to be studied.

7.REFERENCES

- [1] D. Austin,(2017) How does Google find your needle in the haystack of the web, <http://www.ams.org/samplings/feature-column/fcarc-pagerank>.
- [2] Cybrary. IT(2017), Shodan: the hacker's search engine, <https://www.cybrary.it/Op3n/intro-shodansearch-engine-hackers/>
- [3] Rungsawang A, AngKawattanawit N. (2005) Learnable topic-specific web crawler. Journal of Network&Computer Applications, 28(2):97-114.
- [4] Kozanidis L. (2008) An Ontology-Based Focused Crawler. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 376-379.
- [5] P.Pantel, D, Lin,(2002), Document clustering with committees, SIGIR Forum (ACM Special Interest Group on Information Retrieval):199-206.
- [6] Deka, GC, (2018) NoSQL Web Crawler Application. Advances in Computers.109:77-100.
- [7] G. Pavai1, T. V. Geetha. (2016) Improving the freshness of the search engines by a probabilistic approach based incremental crawler. Springer Science+Business Media New York.19:1013- 1028
- [8] P. Boldi, B. Codenotti, M. Santini, S. Vigna(2017) UbiCrawler: a scalable fully distributed web crawler, <http://vigna.di.unimi.it/ftp/papers/UbiCrawler.pdf>.
- [9] Sun LW, He GH, Wu LF (2010), Research on web crawler technology, Store brain knowledge and technology,06(15):4112-4115.
- [10] Pan XY, Chen L, Yu HM, Zhao YJ, Xiao KN, (2019) Survey on research of themed crawling technique. Application Research of Computers.Vol.37, No.5

8.APPENDIX

Source Code:

general.py:

```
import os

# Each website is a separate project (folder)
def create_project_dir(directory):    if not
os.path.exists(directory):
    print('Creating directory ' + directory)
os.makedirs(directory)

# Create queue and crawled files (if not created)
def create_data_files(project_name, base_url):
    queue = os.path.join(project_name ,
'queue.txt')    crawled =
os.path.join(project_name, "crawled.txt"
)    if not os.path.isfile(queue):
write_file(queue, base_url)    if not
os.path.isfile(crawled):
write_file(crawled, '')

# Create a new file def
write_file(path, data):
with open(path, 'w') as f:
    f.write(data)

# Add data onto an existing file def
append_to_file(path, data):
with open(path, 'a') as file:
file.write(data + '\n')
```

```

# Delete the contents of a file def
delete_file_contents(path):
    open(path, 'w').close()

# Read a file and convert each line to set items def
file_to_set(file_name):
    results = set()
    with open(file_name, 'rt') as f:
        for line in f:
            results.add(line.replace('\n', ''))
    return results

# Iterate through a set, each item will be a line
in a file def set_to_file(links, file_name):
    with open(file_name, "w") as f:
        for l in sorted(links):
            f.write(l+"\n")

```

link_finder.py:

```

from html.parser import HTMLParser
from urllib import parse

class LinkFinder(HTMLParser):
    def __init__(self, base_url,
page_url):
        super().__init__()
        self.base_url = base_url
        self.page_url = page_url
        self.links = set()

    # When we call HTMLParser feed() this function
    is called when it encounters an opening tag <a>
    def handle_starttag(self, tag, attrs):

```

```

        if tag == 'a':
            for
            (attribute, value) in attrs:
            if attribute == 'href':
            url = parse.urljoin(self.base_url, value)
            self.links.add(url)
            def
            page_links(self):
            return self.links
            def error(self,
            message):
            pass

```

domain.py:

```

from urllib.parse import urlparse

# Get domain name (example.com) def
get_domain_name(url):
    try:
        results =
        get_sub_domain_name(url).split('.')
        return results[-2] + '.' + results[-1]
    except:
        return ''

# Get sub domain name (name.example.com) def
get_sub_domain_name(url):
    try:
        return urlparse(url).netloc
    except:
        return ''

```

spider.py:

```

from urllib.request import urlopen
from link_finder import LinkFinder
from domain import * from general
import *
class
Spider:
    project_name =
    '        base_url = ''
    domain_name = ''
    queue_file = ''
    crawled_file = ''
    queue = set()
    crawled = set()
    def __init__(self, project_name,
base_url, domain_name):
        Spider.project_name = project_name
        Spider.base_url = base_url
        Spider.domain_name = domain_name
        Spider.queue_file = Spider.project_name +
'/queue.txt'
        Spider.crawled_file = Spider.project_name +
'/crawled.txt'
    self.boot()
        self.crawl_page('First spider',
Spider.base_url)

    # Creates directory and files for project on
    first run and starts the spider
    @staticmethod
    def boot():
        create_project_dir(Spider.project_name)
        create_data_files(Spider.project_name,
Spider.base_url)
        Spider.queue =
        file_to_set(Spider.queue_file)
        Spider.crawled =
        file_to_set(Spider.crawled_file)

```

```

        # Updates user display, fills queue and updates
files
        @staticmethod      def
crawl_page(thread_name, page_url):
if page_url not in Spider.crawled:
    print(thread_name + ' now crawling ' +
page_url)
        print('Queue ' + str(len(Spider.queue))
+ ' | Crawled ' + str(len(Spider.crawled)))

Spider.add_links_to_queue(Spider.gather_links(page_
url))
        Spider.queue.remove(page_url)
        Spider.crawled.add(page_url)
        Spider.update_files()

        # Converts raw response data into readable
information and checks for proper html formatting
        @staticmethod      def
gather_links(page_url):
        html_string = ''
try:
        response = urlopen(page_url)
if 'text/html' in
response.getheader('Content-Type'):
        html_bytes = response.read()
html_string = html_bytes.decode("utf-8")
        finder = LinkFinder(Spider.base_url,
page_url)
        finder.feed(html_string)
except Exception as e:
        print(str(e))
return set()          return
finder.page_links()

        # Saves queue data to project files
        @staticmethod      def
add_links_to_queue(links):
for url in links:

```

```

        if (url in Spider.queue) or (url in
Spider.crawled):
            continue
if Spider.domain_name !=
get_domain_name(url):
            continue
Spider.queue.add(url)

    @staticmethod
def update_files():
    set_to_file(Spider.queue, Spider.queue_file)
    set_to_file(Spider.crawled,
Spider.crawled_file)

```

main.py:

```

import threading from
queue import Queue from
spider import Spider from
domain import * from
general import *

PROJECT_NAME = 'CODETANTRA'
HOMEPAGE =
'https://vitcc.codetantra.com/secure/home.jsp'
DOMAIN_NAME = get_domain_name(HOMEPAGE)
QUEUE_FILE = PROJECT_NAME + '/queue.txt'
CRAWLED_FILE = PROJECT_NAME + '/crawled.txt'
NUMBER_OF_THREADS = 16 queue
= Queue()
Spider(PROJECT_NAME, HOMEPAGE, DOMAIN_NAME)

# Create worker threads (will die when main exits)
def create_workers():
    for _ in range(NUMBER_OF_THREADS):
t = threading.Thread(target=work)
        t.daemon = True

```



```

t.start()

# Do the next job in the queue
def work():    while True:
    url = queue.get()

    Spider.crawl_page(threading.current_thread().name,
url)

    queue.task_done()

# Each queued link is a new job def
create_jobs():    for link in
file_to_set(Queue_FILE):
    queue.put(link)
queue.join()
crawl()

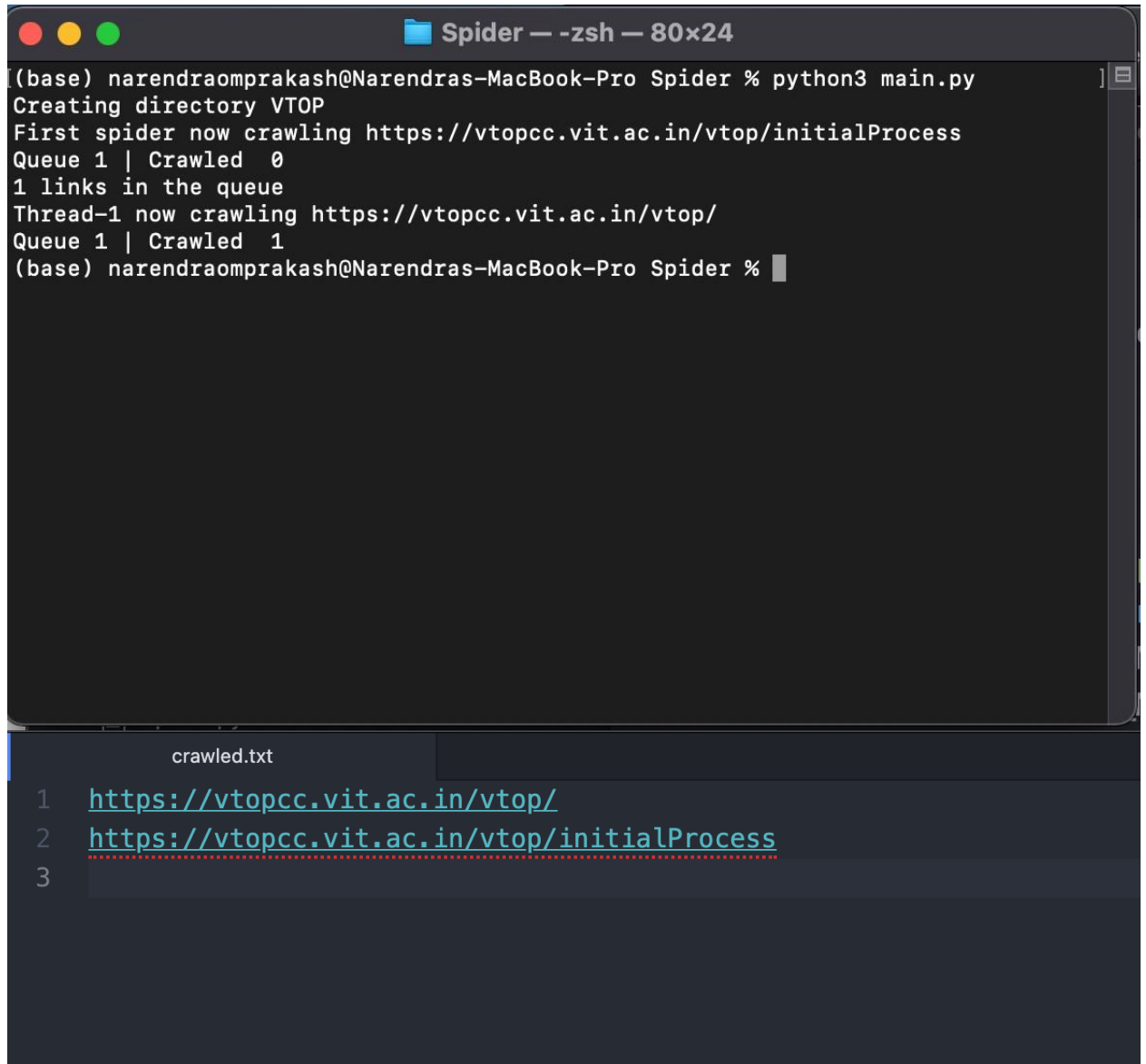
# Check if there are items in the queue, if so
crawl them def crawl():
    queued_links = file_to_set(Queue_FILE)
if len(queued_links) > 0:
    print(str(len(queued_links)) + ' links in
the queue')
    create_jobs()

create_workers() crawl()

```

Sample Output:

For VTOP:



```
Spider — -zsh — 80x24
(base) narendraomprakash@Narendras-MacBook-Pro Spider % python3 main.py
Creating directory VTOP
First spider now crawling https://vtopcc.vit.ac.in/vtop/initialProcess
Queue 1 | Crawled 0
1 links in the queue
Thread-1 now crawling https://vtopcc.vit.ac.in/vtop/
Queue 1 | Crawled 1
(base) narendraomprakash@Narendras-MacBook-Pro Spider %
```



```
crawled.txt
1 https://vtopcc.vit.ac.in/vtop/
2 https://vtopcc.vit.ac.in/vtop/initialProcess
3 .....
```

For CodeTantra:

```
Spider — -zsh — 80x24

(base) narendraomprakash@Narendras-MacBook-Pro Spider % python3 main.py
Creating directory CODETANTRA
First spider now crawling https://vitcc.codetantra.com/secure/home.jsp
Queue 1 | Crawled 0
3 links in the queue
Thread-1 now crawling https://auth.codetantra.com/a/d.jsp?c=y
Queue 3 | Crawled 1
Thread-2 now crawling https://vitcc.codetantra.com/
Thread-3 now crawling https://vitcc.codetantra.com/privacy-policy.jsp
Queue 3 | Crawled 1
Queue 3 | Crawled 1
19 links in the queue
Thread-4 now crawling https://vitcc.codetantra.com/contact-us.jsp
Thread-5 now crawling https://vitcc.codetantra.com/hadoop-course-details.jsp
Queue 19 | Crawled 4
Thread-8 now crawling https://vitcc.codetantra.com/#LiveMeetings
Thread-12 now crawling https://vitcc.codetantra.com/python-programing-course-det
ails.jsp
Thread-13 now crawling https://vitcc.codetantra.com/about-us.jsp
Thread-15 now crawling https://vitcc.codetantra.com/#Customers
Thread-1 now crawling https://vitcc.codetantra.com/cpp-programing-course-details
.jsp
Thread-2 now crawling https://vitcc.codetantra.com/secure/home.jsp#Recognitions
Thread-7 now crawling https://vitcc.codetantra.com/secure/login.jsp

crawled.txt
1 https://auth.codetantra.com/a/d.jsp?c=y
2 https://learn.codetantra.com/buy.jsp
3 https://learn.codetantra.com/buy.jsp?cat=57bc61c70cf269aed3c69da5&cyCode=USD
4 https://vitcc.codetantra.com/
5 https://vitcc.codetantra.com/#Customers
6 https://vitcc.codetantra.com/#LiveMeetings
7 https://vitcc.codetantra.com/#OnlineCourses
8 https://vitcc.codetantra.com/#Recognitions
9 https://vitcc.codetantra.com/#RemoteProctoring
10 https://vitcc.codetantra.com/#VirtualX
11 https://vitcc.codetantra.com/123.jsp
12 https://vitcc.codetantra.com/about-us.jsp
13 https://vitcc.codetantra.com/c-programing-course-details.jsp
14 https://vitcc.codetantra.com/contact-us.jsp
15 https://vitcc.codetantra.com/cpp-programing-course-details.jsp
16 https://vitcc.codetantra.com/few-java-practice-exercises-from-tutorial.jsp
17 https://vitcc.codetantra.com/hadoop-course-details.jsp
18 https://vitcc.codetantra.com/java-course-details.jsp
19 https://vitcc.codetantra.com/java/jdk6.0/tutorial/essential/concurrency/runthread.html
20 https://vitcc.codetantra.com/java/jdk6.0/tutorial/essential/concurrency/simple.html
21 https://vitcc.codetantra.com/java/jdk6.0/tutorial/essential/concurrency/sleep.html
22 https://vitcc.codetantra.com/java/jdk6.0/tutorial/essential/io/bytestreams.html
23 https://vitcc.codetantra.com/java/jdk6.0/tutorial/essential/io/charstreams.html
24 https://vitcc.codetantra.com/java/jdk6.0/tutorial/essential/io/formatting.html
25 https://vitcc.codetantra.com/java/jdk6.0/tutorial/essential/io/scanning.html
26 https://vitcc.codetantra.com/java/jdk6.0/tutorial/java/landI/override.html
27 https://vitcc.codetantra.com/java/jdk6.0/tutorial/java/landI/super.html
28 https://vitcc.codetantra.com/java/jdk6.0/tutorial/java/data/beyondmath.html
29 https://vitcc.codetantra.com/java/jdk6.0/tutorial/java/data/buffers.html
30 https://vitcc.codetantra.com/java/jdk6.0/tutorial/java/data/comparestrings.html
31 https://vitcc.codetantra.com/java/jdk6.0/tutorial/java/data/converting.html
```