

WEB CRAWLER FOR EBOOK LIBRARY

BY

ABHISHEK KUMAR 101336

PROJECT SUPERVISOR: MR. SUMAN SAHA



MAY -2014

Submitted in partial fulfillment of the Degree of

Bachelor of Technology

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING AND
INFORMATION TECHNOLOGY**

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,

WAKNAGHAT

Table of Contents

Certificate from Supervisor	I
Acknowledgement	II
Abstract	III
List of Figures	IV
Chapter 1 : INTRODUCTION	
1.1 Overview	2
1.2 Problem Statement	3
1.3 Motivation	3
1.4 Objective	3
Chapter 2 : LITRATURE	
2.1 Web Crawler	5
2.2 Focused Crawler	6
2.3 Indexing	7
2.4 Page Ranking	8
Chapter 3 : CRAWLING	
3.1 Procedure	11
3.2 Crawling Issues	
3.2.1 Robots.txt	12
3.2.2 Spider Trap	13
3.2.3 Preprocessing	
3.2.3.1 Stop-Word-Removal	13
3.2.3.2 Stemming	14
Chapter 4 : CODE IMPLEMENTATION	
4.1 Code for eBook Crawler	16
4.2 Output	40
CONCLUSION	42
Bibliography	43

(1)

CERTIFICATE

This is to certify that the work titled “**WEB CRAWLER FOR EBOOK LIBRARY**“ Submitted by **ABHISHEK KUMAR** in partial fulfillment for the award of degree of B.Tech Computer Science Engineering of Jaypee University of Information Technology, Waknaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

(Signature of Supervisor)

Name of Supervisor: Mr. Suman Saha

Designation: Asst. Professor, Dept. of CSE and ICT

Date: 15th May 2014

(II)

ACKNOWLEDGEMENT

We owe a great thanks to many people who have been helping and supporting us during this project. Our heartiest thanks to **Mr. Suman Saha**, the Project Guide for guiding and correcting us at every step of our work with attention and care. He has taken pain to go through the project and make necessary correction as and when needed. Thanks and appreciation to the helpful people at college for their support. We would also thank our university and my faculty members without whom this project would have been a distant reality. We also extend our heartfelt thanks to our family and well-wishers.

Signature of Student:

Name of Student: Abhishek Kumar (101336)

Date: 15th May 2014

(III)

ABSTRACT

A crawler is a program that retrieves and stores pages from the Web, commonly for a Web search engine. A crawler often has to download hundreds of millions of pages in a short period of time and has to constantly monitor and refresh the downloaded pages. A focused crawler is a Web crawler aiming to search and retrieve Web pages from the World Wide Web, which are related to a domain-specific topic. Rather than downloading all accessible Web pages, a focused crawler analyzes the frontier of the crawled region to visit only the portion of the Web that contains relevant Web pages, and at the same time, try to skip irrelevant regions.

A web crawler for ebook library is a web crawler which replies to ebook related queries. The crawler for ebook Library must crawl through the ebook specific Web pages in the World Wide Web (WWW). For a crawler it is not an easy task to download the ebook specific Web pages. Focus Crawling Mechanism can play a vital role in this context. In our approach we crawl through the Web and store eBooks in a fileSystem, which are related to a eBook domain using Focus Crawling Mechanism.

(IV)

List of Figures

Fig 2.1 - Working of General Crawler	5
Fig 2.2 - Working of Focused Crawler	7
Fig 4.1 - Output of eBook Crawler	40
Fig 4.2 – Log file of Output	41

Chapter 1: INTRODUCTION

1.1 Overview

A Web crawler is a program that downloads Web pages, commonly for a Web search engine or a Web cache. Roughly, a crawler starts off with an initial set of URLs. It first URL in a queue, where all URLs to be retrieved are kept and prioritized. From this queue, the crawler gets a URL (in some order), downloads the page, extracts any URLs in the downloaded page, and puts the new URLs in the queue. This process is repeated until the crawler decides to stop, for any one of various reasons. Every page that is retrieved is given to a client that saves the pages, creates an index for the pages, or analyzes the content of the pages.

Crawlers are widely used today. Crawlers for the major search engines (e.g., Google, Yahoo, and Bing) attempt to visit a significant portion of textual Web pages, in order to build content indexes. Other crawlers may also visit many pages, but may look only for certain types of information.

Web Crawler for eBook Library use focused crawlers to selectively collect Web pages relevant to particular ebook domains such that they can keep smaller Web page collections and, at the same time, provide search results with high precision. Unlike general-purpose Web crawler which automatically traverses the Web and collects all Web, focused crawling is designed to gather collection of pages on specific topic. A focused crawler tries to “predict” whether or not a target URL is pointing to a relevant and high-quality Web page before actually fetching the page, and then it follows the most appropriate links, leading to retrieval of more relevant pages and greater saves in resources. In this crawler it traverse only those links or paths within a specific domain : eBook. In this crawler it crawls the eBook available in web and store in a fileSystem.

1.2 Problem Statement

I interested to design a custom web crawler to search online eBook available in the Web. We can use it as a search engine for eBook and crawl through the Web and store eBooks in a fileSystem, which are related to a eBook domain .

1.3 Motivation

I highly motivated to this topic because currently and in the future, this project will be use very frequently because today's world we use advance technology(eg. Mobile , TabletPC, Phablets etc) where people want to get all needs in its finger and students prefers to download eBook instead of buying the hard copy of book and spending lots of money on to it. In other hand if we talk about the general crawler which crawl the complete web that not give the very relevant result which we want, they give the complete result of our query and we have to search on them So this crawler gives the result related to ebook only.

1.4 Objectives

The goal of this project is to build a custom web "spider" or "crawler" which is search online eBook and traverse links related to ebook domain available in Web.

Chapter 2 : LITERATURE REVIEW

2.1 Web Crawler

A **crawler** is a program that visits Web sites and reads their pages and other information in order to create entries for a search engine index. The major search engines on the Web all have such a program, which is also known as a "spider" or a "bot." Crawlers are typically programmed to visit sites that have been submitted by their owners as new or updated. Entire sites or specific pages can be selectively visited and indexed. Crawlers apparently gained the name because they crawl through a site a page at a time, following the links to other pages on the site until all pages have been read.

A web Crawler first create a list of URLs and then visits these URLs , called seeds. When crawler visits all URLs which is in the list , it identifies the hyperlinks available in the pages and add them to visiting URLs list, called the crawl frontier. URLs from the frontiers follows some set of policies to visiting the URLs. If the crawler is crawling the website then it copies and saves the information and stored such that then can viewed, read and navigated on the live web. In a certain given time crawler download a limited number of Web Pages which implies large volume So, its need to prioritize its downloads..

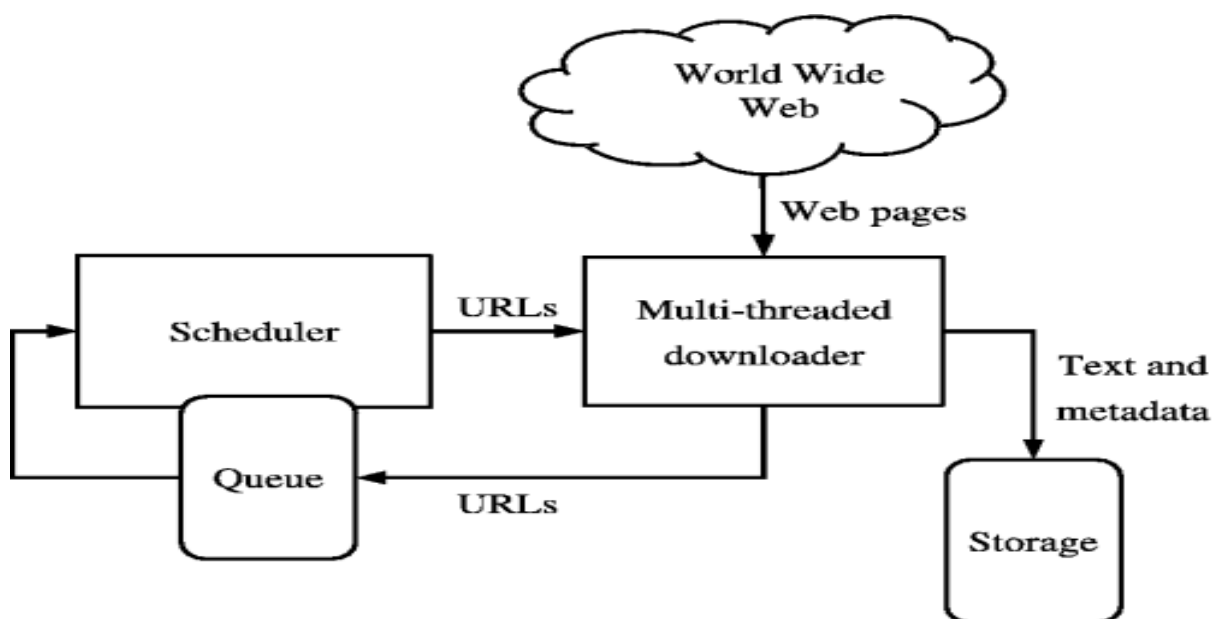


Figure 2.1- Working of General Crawler

2.2 Focused Crawler

A **Focused crawler** is a web crawler that attempts to download only web pages that are relevant to our topic. A focused crawler is a web crawler that collects Web pages that satisfy some specific property, by carefully prioritizing the crawl frontier and managing the hyperlink exploration process. Some predicates may be based on simple, deterministic and surface properties. For example, a crawler's mission may be to crawl pages from only the .jp domain. Other predicates may be softer or comparative, e.g., "crawl pages with large PageRank", or "crawl pages about baseball". An important page property pertains to topics, leading to topical crawlers. For example, a topical crawler may be deployed to collect pages about solar power, or swine flu, while minimizing resources spent fetching pages on other topics. Crawl frontier management may not be the only device used by focused crawlers; they may use a Web directory, an Web text index, backlinks, or any other Web artifact. A focused crawler must predict the probability that an unvisited page will be relevant before actually downloading the page. Crawlers are also focused on page properties other than topics.

The performance of the focused crawler depends on the richness of URLs in the relevant topic being searched and it relies normal search engine for providing the starting points. Selection of seed URLs is very important for focused crawler and crawling efficiency. Focused crawler start the focus crawl from a list of high quality seed URLs and limit the crawling scope to the domain of these URLs and this list of high quality URLs strategy are a Whitelist Strategy. The Whitelist high quality seeds should be selected based on a list of URLs which are accumulated over a long period of normal web crawling. The list of high quality should be updated time to time after it is created. The advantage of focused crawler are it take less time to give relevant result and you spend less effort on processing web pages and less money that are unlikely to be value . It basically takes less time for crawl as compare to general crawler and search our related topic and search high value pages related to our topic.

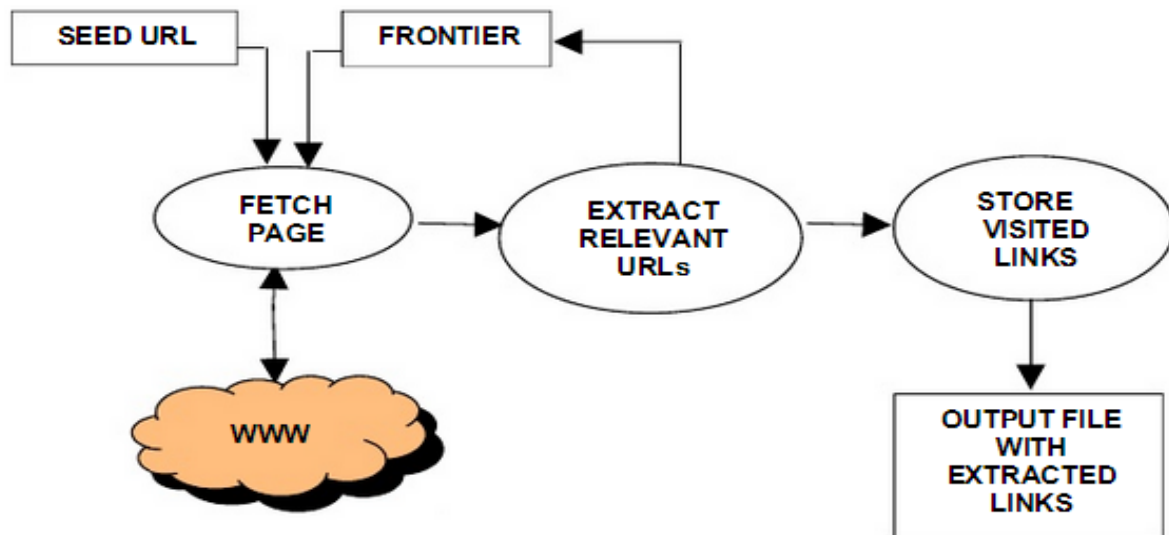


Figure 2.2 Working of Focused Crawler

2.3 Indexing

Tokenization is the process of breaking up an entire document into individual, indexable keywords. The first process is to remove all HTML markup from the document. Because HTML is a tag-based language, we should remove the text within the tags; in this way, we avoid indexing the word "body" as found in "<body>". In certain cases, we should also dismiss the information found in between opening and closing tags. For instance, we are probably not interested in indexing the client-side scripting available on the site (generally, Javascript). Thus we disregard all text found in between the <script> and </script> tags. Other tags (such as the <body> tag) contain very useful information between their opening and closing and should be kept.

For a Web index, one solution is that those Web pages should come from as many different servers as possible. The Web Crawler takes the following approach: it uses a modified breadth-first algorithm to ensure that every server has at least one Web page represented in the index. This strategy is very effective. In detail, a WebCrawler indexing run proceeds as follows: every time a Web page on a new server is found, that server is placed on a list of servers to be visited right away. Before any other Web pages are visited, a Web page on each of the new servers is retrieved and indexed.

When all known servers have been visited, indexing proceeds sequentially through a list of all servers until a new one is found, at which point the process repeats. During indexing, the WebCrawler runs either for a certain amount of time, or until it has retrieved some number of Web pages. The basic idea of following different Web pages from one to another using the links came was first demonstrated to work in the search. The Web Crawler extends that concept to initiate the search using the index, and to follow links in an intelligent order.

Googlebot processes each of the pages it crawls in order to compile a massive index of all the words it sees and their location on each page. In addition, we process information included in key content tags and attributes, such as Title tags and ALT attributes. Googlebot can process many, but not all, content types. For example, we cannot process the content of some rich media files or dynamic pages.

2.4 Page Ranking

PageRank is a link analysis algorithm and it assigns a numerical weighting to each element of a hyperlinked set of documents, such as the World Wide Web, with the purpose of "measuring" its relative importance within the set. The algorithm may be applied to any collection of entities with reciprocal quotations and references. The numerical weight that it assigns to any given element E is referred to as the PageRank of E and denoted by $PR(E)$. Other factors like Author Rank can contribute to the importance of an entity. A PageRank results from a mathematical algorithm based on the webgraph, created by all World Wide Web pages as nodes and hyperlinks as edges, taking into consideration authority hubs such as cnn.com or usa.gov. The rank value indicates an importance of a particular page. A hyperlink to a page counts as a vote of support. The PageRank of a page is defined recursively and depends on the number and PageRank metric of all pages that link to it ("incoming links"). A page that is linked to by many pages with high PageRank receives a high rank itself.

PageRank is the measure of the importance of a page based on the incoming links from other pages. In simple terms, each link to a page on your site from another site adds to your site's PageRank. Not all links are equal: Google works hard to improve

the user experience by identifying spam links and other practices that negatively impact search results. The best types of links are those that are given based on the quality of your content.

Googlebot processes each of the pages it crawls in order to compile a massive index of all the words it sees and their location on each page. In addition, we process information included in key content tags and attributes, such as Title tags and ALT attributes. Googlebot can process many, but not all, content types. For example, we cannot process the content of some rich media files or dynamic pages.

PageRank is a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. PageRank can be calculated for collections of documents of any size. It is assumed in several research papers that the distribution is evenly divided among all documents in the collection at the beginning of the computational process. The PageRank computations require several passes, called "iterations", through the collection to adjust approximate PageRank values to more closely reflect the theoretical true value.

A probability is expressed as a numeric value between 0 and 1. A 0.5 probability is commonly expressed as a "50% chance" of something happening. Hence, a PageRank of 0.5 means there is a 50% chance that a person clicking on a random link will be directed to the document with the 0.5 PageRank.

Chapter 3: CRAWLING

3.1 Procedure

Procedure for detect and extract links from pages is as follows:

1. Start at the <body> tag if one exists. This presumes that there is no page content stored in the HTML head.
2. Find the next instance of "<a" (ignoring case), as links begin with an "<a href> tag.
3. Check to see if "href" and "=" are in this same tag, if so, continue.
4. Find the quotation mark immediately following the "href" and the "=". Extract the link enclosed in these quotations.
5. Trim a "#" or "?" and the subsequent text off from the end of the link.
6. Trim "index.html" off of the end of the string, if present.
7. Convert all non-alpha-numeric (excluding "/" and ":") to their hexadecimal equivalents (spaces convert to "%20", etc.
8. Recall the origin URL from which this link is being extracted (in case of relative links).
9. For every instance of "../" in the link URL, trim the last directory off of the base URL (to ascend into its parent directory), and trim an instance of "../" from the link. Repeat until either there are no more "../"s or until the parent directory ends in ".edu", indicating that the highest parent URL is the base.
10. Trim "www." off of the front, if it is present.
11. Check to see if the link already exists in the database. If so, add a reference from this page pointing to the pre-existing link so that we know that it links to that page. If not, add that link to the list of all URLs in the domain, and add a reference from this page to that one.
12. Go back to step 2, if applicable.

3.2 Crawling Issues

3.2.1 robots.txt

The Robot Exclusion Standard, also known as the Robots Exclusion Protocol or robots.txt protocol, is a convention to advising cooperating web crawlers and other web robots about accessing all or part of a website which is otherwise publicly viewable. Robots are often used by search engines to categorize and archive web sites, or by webmasters to proofread source code. The standard is different from, but can be used in conjunction with, Sitemaps, a robot inclusion standard for websites.

A robots.txt file on a website will function as a request that specified robots ignore specified files or directories when crawling a site. This might be, for example, out of a preference for privacy from search engine results, or the belief that the content of the selected directories might be misleading or irrelevant to the categorization of the site as a whole, or out of a desire that an application only operate on certain data. Links to pages listed in robots.txt can still appear in search results if they are linked to from a page that is crawled.

A robots.txt file covers one origin. For websites with multiple subdomains, each subdomain must have its own robots.txt file. If example.com had a robots.txt file but a.example.com did not, the rules that would apply for example.com would not apply to a.example.com. In addition, each protocol and port needs its own robots.txt file; <http://example.com/robots.txt> does not apply to pages under <https://example.com:8080/> or <https://example.com/>.

Example of robots.txt:

```
user-agent: *  
  
Disallow: /cgi-bin/  
  
Disallow: /registration/
```

The "User-agent: *" means this section applies to all robots. The "Disallow: /" tells the robot that it should not visit any pages on the site.

Robots can ignore your /robots.txt. Especially malware robots that scan the web for security vulnerabilities, and email address harvesters used by spammers will pay no attention.

3.2.2 Spider trap

A spider trap (or crawler trap) is a set of web pages that may intentionally or unintentionally be used to cause a web crawler or search bot to make an infinite number of requests or cause a poorly constructed crawler to crash. Web crawlers are also called web spiders, from which the name is derived. Spider traps may be created to "catch" spambots or other crawlers that waste a website's bandwidth. They may also be created unintentionally by calendars that use dynamic pages with links that continually point to the next day or year.

Example of creation of indefinitely deep directory structures like :

<http://abc.com/a/b/c/b/c/a/b/a/b/c/a/b/.....>

3.2.3 Preprocessing

3.2.3.1 Stop-Word-Removal

Common words that carry less important meaning than the actual keywords. Stop-words are a list of words which are viewed as unhelpful in narrowing down searches. Words like "the" are generally not relevant to a query or to the ideal results of a query (unless, of course, a document is about the word "The." Such exceptions should be considered in a real-world IR system). These words are then excluded from the index. stop words are words which are filtered out prior to, or after, processing of natural language data (text). There is not one definite list of stop words which all tools use and such a filter is not always used. Some tools specifically avoid removing them to support phrase search.

For example of some stop words: a, the, this, etc.

3.2.3.2 Stemming

The process for reducing inflected word to their stem, base or root form. Stemming is the practice of reducing a word to its stem or root. The goal of stemming is to improve information-retrieval on a document by "clustering" similar words together. In this way, searches can be performed on the clusters of similar words as opposed to each variant of the word individually.

For example: caresses --> caress

 Cats --> cat

Chapter 4: CODE IMPLEMENTATION

4.1 Code for eBook Crawler

```
import java.awt.*;

import java.awt.event.*;

import java.io.*;

import java.net.*;

import java.util.*;

import java.util.regex.*;

import javax.swing.*;

import javax.swing.table.*;

public class EbookCrawler extends JFrame
{
    private static final String[] MAX_URLS = {"50", "100", "500", "1000"};

    private HashMap disallowListCache = new HashMap();

    private JTextField startTextField;

    private JComboBox maxComboBox;

    private JCheckBox limitCheckBox;

    private JTextField logTextField;

    private JTextField searchTextField;

    private JCheckBox caseCheckBox;

    private JButton searchButton;

    private JLabel crawlingLabel2;

    private JLabel crawledLabel2;
```

```
private JLabel toCrawlLabel2;

private JProgressBar progressBar;

private JLabel matchesLabel2;

private JTable table;

private boolean crawling;

private PrintWriter logFileWriter;

public EbookCrawler()

{

setTitle("Ebook Crawler");

setSize(600, 600);

addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent e) {

actionExit();

}

});

JMenuBar menuBar = new JMenuBar();

JMenu fileMenu = new JMenu("File");

fileMenu.setMnemonic(KeyEvent.VK_F);

JMenuItem fileExitMenuItem = new JMenuItem("Exit",KeyEvent.VK_X);

fileExitMenuItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

actionExit();

}

}
```

```
});  
  
fileMenu.add(fileExitMenuItem);  
  
menuBar.add(fileMenu);  
  
setJMenuBar(menuBar);  
  
JPanel searchPanel = new JPanel();  
  
GridBagConstraints constraints;  
  
GridBagLayout layout = new GridBagLayout();  
  
searchPanel.setLayout(layout);  
  
JLabel startLabel = new JLabel("Start URL:");  
  
constraints = new GridBagConstraints();  
  
constraints.anchor = GridBagConstraints.EAST;  
  
constraints.insets = new Insets(5, 5, 0, 0);  
  
layout.setConstraints(startLabel, constraints);  
  
searchPanel.add(startLabel);  
  
startTextField = new JTextField();  
  
constraints = new GridBagConstraints();  
  
constraints.fill = GridBagConstraints.HORIZONTAL;  
  
constraints.gridwidth = GridBagConstraints.REMAINDER;  
  
constraints.insets = new Insets(5, 5, 0, 5);  
  
layout.setConstraints(startTextField, constraints);  
  
searchPanel.add(startTextField);  
  
JLabel maxLabel = new JLabel("Max URLs to Crawl:");  
  
constraints = new GridBagConstraints();
```



```
constraints.anchor = GridBagConstraints.EAST;

constraints.insets = new Insets(5, 5, 0, 0);

layout.setConstraints(maxLabel, constraints);

searchPanel.add(maxLabel);

maxComboBox = new JComboBox(MAX_URLS);

maxComboBox.setEditable(true);

constraints = new GridBagConstraints();

constraints.insets = new Insets(5, 5, 0, 0);

layout.setConstraints(maxComboBox, constraints);

searchPanel.add(maxComboBox);

limitCheckBox = new JCheckBox("Limit crawling to Start URL site");

constraints = new GridBagConstraints();

constraints.anchor = GridBagConstraints.WEST;

constraints.insets = new Insets(0, 10, 0, 0);

layout.setConstraints(limitCheckBox, constraints);

searchPanel.add(limitCheckBox);

JLabel blankLabel = new JLabel();

constraints = new GridBagConstraints();

constraints.gridwidth = GridBagConstraints.REMAINDER;

layout.setConstraints(blankLabel, constraints);

searchPanel.add(blankLabel);

JLabel logLabel = new JLabel("Matches Log File:");

constraints = new GridBagConstraints();
```

```
constraints.anchor = GridBagConstraints.EAST;

constraints.insets = new Insets(5, 5, 0, 0);

layout.setConstraints(logLabel, constraints);

searchPanel.add(logLabel);

String file = System.getProperty("user.dir") + System.getProperty("file.separator") +
"crawler.log";

logTextField = new JTextField(file);

constraints = new GridBagConstraints();

constraints.fill = GridBagConstraints.HORIZONTAL;

constraints.gridwidth = GridBagConstraints.REMAINDER;

constraints.insets = new Insets(5, 5, 0, 5);

layout.setConstraints(logTextField, constraints);

searchPanel.add(logTextField);

JLabel searchLabel = new JLabel("Search String:");

constraints = new GridBagConstraints();

constraints.anchor = GridBagConstraints.EAST;

constraints.insets = new Insets(5, 5, 0, 0);

layout.setConstraints(searchLabel, constraints);

searchPanel.add(searchLabel);

searchTextField = new JTextField();

constraints = new GridBagConstraints();

constraints.fill = GridBagConstraints.HORIZONTAL;

constraints.insets = new Insets(5, 5, 0, 0);
```

```
constraints.gridwidth= 2;

constraints.weightx = 1.0d;

layout.setConstraints(searchTextField, constraints);

searchPanel.add(searchTextField);

caseCheckBox = new JCheckBox("Case Sensitive");

constraints = new GridBagConstraints();

constraints.insets = new Insets(5, 5, 0, 5);

constraints.gridwidth = GridBagConstraints.REMAINDER;

layout.setConstraints(caseCheckBox, constraints);

searchPanel.add(caseCheckBox);

searchButton = new JButton("Search");

searchButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

actionSearch();

}

});

constraints = new GridBagConstraints();

constraints.gridwidth = GridBagConstraints.REMAINDER;

constraints.insets = new Insets(5, 5, 5, 5);

layout.setConstraints(searchButton, constraints);

searchPanel.add(searchButton);

JSeparator separator = new JSeparator();

constraints = new GridBagConstraints();
```

```
constraints.fill = GridBagConstraints.HORIZONTAL;

constraints.gridwidth = GridBagConstraints.REMAINDER;

constraints.insets = new Insets(5, 5, 5, 5);

layout.setConstraints(separator, constraints);

searchPanel.add(separator);

JLabel crawlingLabel1 = new JLabel("Crawling:");

constraints = new GridBagConstraints();

constraints.anchor = GridBagConstraints.EAST;

constraints.insets = new Insets(5, 5, 0, 0);

layout.setConstraints(crawlingLabel1, constraints);

searchPanel.add(crawlingLabel1);

crawlingLabel2 = new JLabel();

crawlingLabel2.setFont(
crawlingLabel2.getFont().deriveFont(Font.PLAIN));

constraints = new GridBagConstraints();

constraints.fill = GridBagConstraints.HORIZONTAL;

constraints.gridwidth = GridBagConstraints.REMAINDER;

constraints.insets = new Insets(5, 5, 0, 5);

layout.setConstraints(crawlingLabel2, constraints);

searchPanel.add(crawlingLabel2);

JLabel crawledLabel1 = new JLabel("Crawled URLs:");

constraints = new GridBagConstraints();

constraints.anchor = GridBagConstraints.EAST;
```

```
constraints.insets = new Insets(5, 5, 0, 0);  
layout.setConstraints(crawledLabel1, constraints);  
searchPanel.add(crawledLabel1);  
crawledLabel2 = new JLabel();  
crawledLabel2.setFont(  
crawledLabel2.getFont().deriveFont(Font.PLAIN));  
constraints = new GridBagConstraints();  
constraints.fill = GridBagConstraints.HORIZONTAL;  
constraints.gridwidth = GridBagConstraints.REMAINDER;  
constraints.insets = new Insets(5, 5, 0, 5);  
layout.setConstraints(crawledLabel2, constraints);  
searchPanel.add(crawledLabel2);  
JLabel toCrawlLabel1 = new JLabel("URLs to Crawl:");  
constraints = new GridBagConstraints();  
constraints.anchor = GridBagConstraints.EAST;  
constraints.insets = new Insets(5, 5, 0, 0);  
layout.setConstraints(toCrawlLabel1, constraints);  
searchPanel.add(toCrawlLabel1);  
toCrawlLabel2 = new JLabel();  
toCrawlLabel2.setFont(  
toCrawlLabel2.getFont().deriveFont(Font.PLAIN));  
constraints = new GridBagConstraints();  
constraints.fill = GridBagConstraints.HORIZONTAL;
```

```
constraints.gridwidth = GridBagConstraints.REMAINDER;

constraints.insets = new Insets(5, 5, 0, 5);

layout.setConstraints(toCrawlLabel2, constraints);

searchPanel.add(toCrawlLabel2);

JLabel progressLabel = new JLabel("Crawling Progress:");

constraints = new GridBagConstraints();

constraints.anchor = GridBagConstraints.EAST;

constraints.insets = new Insets(5, 5, 0, 0);

layout.setConstraints(progressLabel, constraints);

searchPanel.add(progressLabel);

progressBar = new JProgressBar();

progressBar.setMinimum(0);

progressBar.setStringPainted(true);

constraints = new GridBagConstraints();

constraints.fill = GridBagConstraints.HORIZONTAL;

constraints.gridwidth = GridBagConstraints.REMAINDER;

constraints.insets = new Insets(5, 5, 0, 5);

layout.setConstraints(progressBar, constraints);

searchPanel.add(progressBar);

JLabel matchesLabel1 = new JLabel("Search Matches:");

constraints = new GridBagConstraints();

constraints.anchor = GridBagConstraints.EAST;

constraints.insets = new Insets(5, 5, 10, 0);
```

```

layout.setConstraints(matchesLabel1, constraints);

searchPanel.add(matchesLabel1);

matchesLabel2 = new JLabel();

matchesLabel2.setFont(matchesLabel2.getFont().deriveFont(Font.PLAIN));

constraints = new GridBagConstraints();

constraints.fill = GridBagConstraints.HORIZONTAL;

constraints.gridwidth = GridBagConstraints.REMAINDER;

constraints.insets = new Insets(5, 5, 10, 5);

layout.setConstraints(matchesLabel2, constraints);

searchPanel.add(matchesLabel2);

table = new JTable(new DefaultTableModel(new Object[][]{ },new String[]{"URL"})
{
    public boolean isCellEditable(int row, int column)
    {
        return false;
    }
});

JPanel matchesPanel = new JPanel();

matchesPanel.setBorder(BorderFactory.createTitledBorder("Matches"));

matchesPanel.setLayout(new BorderLayout());

matchesPanel.add(new JScrollPane(table),

BorderLayout.CENTER);

getContentPane().setLayout(new BorderLayout());

```

```
getContentPane().add(searchPanel, BorderLayout.NORTH);

getContentPane().add(matchesPanel, BorderLayout.CENTER);

}

private void actionExit() {

System.exit(0);

}

private void actionSearch() {

    if (crawling) {

crawling = false;

return;

}

ArrayList errorList = new ArrayList();

String startUrl = startTextField.getText().trim();

if (startUrl.length() < 1) {

errorList.add("Missing Start URL.");

}

else if (verifyUrl(startUrl) == null) {

errorList.add("Invalid Start URL.");

}

int maxUrls = 0;

String max = ((String) maxComboBox.getSelectedItem()).trim();

if (max.length() > 0) {

try {
```



```
maxUrls = Integer.parseInt(max);
}
catch (NumberFormatException e) {
}
if (maxUrls < 1) {
errorList.add("Invalid Max URLs value.");
}
}

String logFile = logTextField.getText().trim();
if (logFile.length() < 1) {
errorList.add("Missing Matches Log File.");
}

String searchString = searchTextField.getText().trim();
if (searchString.length() < 1) {
errorList.add("Missing Search String.");
}

if (errorList.size() > 0) {
StringBuffer message = new StringBuffer();
for (int i = 0; i < errorList.size(); i++) {
message.append(errorList.get(i));
if (i + 1 < errorList.size()) {
message.append("\n");
}
}
```

```

    }

    showError(message.toString());

    return;

}

startUrl = removeWwwFromUrl(startUrl);

search(logFile, startUrl, maxUrls, searchString);

}

private void search(final String logFile, final String startUrl,
    final int maxUrls, final String searchString)
{
    Thread thread = new Thread(new Runnable() {

    public void run() {

        setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));

        startTextField.setEnabled(false);

        maxComboBox.setEnabled(false);

        limitCheckBox.setEnabled(false);

        logTextField.setEnabled(false);

        searchTextField.setEnabled(false);

        caseCheckBox.setEnabled(false);

        searchButton.setText("Stop");

        table.setModel(new DefaultTableModel(new Object[][]{ },
            new String[]{"URL"})) {

        public boolean isCellEditable(int row, int column)

```

```
{  
return false;  
}  
});  
updateStats(startUrl, 0, 0, maxUrls);  
try {  
    logFileWriter = new PrintWriter(new FileWriter(logFile));  
} catch (Exception e) {  
    showError("Unable to open matches log file.");  
    return;  
}  
crawling = true;  
crawl(startUrl, maxUrls, limitCheckBox.isSelected(),  
searchString, caseCheckBox.isSelected());  
crawling = false;  
try {  
    logFileWriter.close();  
} catch (Exception e) {  
    showError("Unable to close matches log file.");  
}  
crawlingLabel2.setText("Done");  
startTextField.setEnabled(true);  
maxComboBox.setEnabled(true);
```

```

limitCheckBox.setEnabled(true);

logTextField.setEnabled(true);

searchTextField.setEnabled(true);

caseCheckBox.setEnabled(true);

searchButton.setText("Search");

setCursor(Cursor.getDefaultCursor());

if (table.getRowCount() == 0) {

JOptionPane.showMessageDialog(EbookCrawler.this,

    "Your Search String was not found. Please try another.",

    "Search String Not Found",

    JOptionPane.WARNING_MESSAGE);

}

}

});

thread.start();

}

private void showError(String message) {

    JOptionPane.showMessageDialog(this, message, "Error",

        JOptionPane.ERROR_MESSAGE);

}

private void updateStats(

    String crawling, int crawled, int toCrawl, int maxUrls)

{

```

```

crawlingLabel2.setText(crawling);

crawledLabel2.setText("" + crawled);

toCrawlLabel2.setText("" + toCrawl);

if (maxUrls == -1) {

progressBar.setMaximum(crawled + toCrawl);

} else {

progressBar.setMaximum(maxUrls);

}

progressBar.setValue(crawled);

matchesLabel2.setText("" + table.getRowCount());

}

private void addMatch(String url) {

DefaultTableModel model =(DefaultTableModel) table.getModel();

model.addRow(new Object[]{url});

try {

logFileWriter.println(url);

} catch (Exception e) {

showError("Unable to log match.");

}

}

private URL verifyUrl(String url) {

if (!url.toLowerCase().startsWith("http://"))

return null;

```

```

URL verifiedUrl = null;

try {

verifiedUrl = new URL(url);

} catch (Exception e) {

return null;

}

return verifiedUrl;

}

boolean isRobotAllowed(URL urlToCheck) {

String host = urlToCheck.getHost().toLowerCase();

ArrayList disallowList = (ArrayList) disallowListCache.get(host);

if (disallowList == null) {

disallowList = new ArrayList();

try {

URL robotsFileUrl =

new URL("http://" + host + "/robots.txt");

BufferedReader reader = new BufferedReader(new InputStreamReader(

robotsFileUrl.openStream()));

String line;

while ((line = reader.readLine()) != null) {

if (line.indexOf("Disallow:") == 0) {

String disallowPath = line.substring("Disallow:".length());

int commentIndex = disallowPath.indexOf("#");

```

```

if (commentIndex != -1) {
disallowPath = disallowPath.substring(0, commentIndex);
}
disallowPath = disallowPath.trim();
disallowList.add(disallowPath);
}
}
disallowListCache.put(host, disallowList);
}
catch (Exception e) {
return true;
}
}
String file = urlToCheck.getFile();
for (int i = 0; i < disallowList.size(); i++) {
String disallow = (String) disallowList.get(i);
if (file.startsWith(disallow)) {
return false;
}
}
return true;
}
private String downloadPage(URL pageUrl) {

```

```

try {
    BufferedReader reader = new BufferedReader(new InputStreamReader(
pageUrl.openStream()));

    String line;

    StringBuffer pageBuffer = new StringBuffer();

    while ((line = reader.readLine()) != null) {
pageBuffer.append(line);
    }

    return pageBuffer.toString();
    } catch (Exception e) {
    }

    return null;
}

private String removeWwwFromUrl(String url) {

    int index = url.indexOf("://www.");

    if (index != -1) {

return url.substring(0, index + 3) +
        url.substring(index + 7);
    }

    return (url);
}

private ArrayList retrieveLinks(URL pageUrl, String pageContents, HashSet

```



```

crawledList, boolean limitHost)
{
    Pattern p
    =Pattern.compile("<a\\s+href\\s*=\\s*\"?(.*?)[\\\">]",Pattern.CASE_INSENSITIVE);

    Matcher m = p.matcher(pageContents);

    ArrayList linkList = new ArrayList();

    while (m.find()) {

        String link = m.group(1).trim();

        if (link.length() < 1) {

            continue;

        }

        if (link.charAt(0) == '#') {

            continue;

        }

        if (link.indexOf("mailto:") != -1) {

            continue;

        }

        if (link.toLowerCase().indexOf("javascript") != -1) {

            continue;

        }

        if (link.indexOf(":/") == -1) {

            if (link.charAt(0) == '/') {

                link = "http://" + pageUrl.getHost() + link;
            }
        }
    }
}

```

```

    } else {
String file = pageUrl.getFile();

if (file.indexOf('/') == -1) {
link = "http://" + pageUrl.getHost() + "/" + link;
} else {
String path = file.substring(0, file.lastIndexOf('/') + 1);
link = "http://" + pageUrl.getHost() + path + link;
}
}
}

int index = link.indexOf('#');
if (index != -1) {
link = link.substring(0, index);
}

link = removeWwwFromUrl(link);
URL verifiedLink = verifyUrl(link);
if (verifiedLink == null) {
continue;
}

if (limitHost &&
!pageUrl.getHost().toLowerCase().equals(verifiedLink.getHost().toLowerCase()))
{
continue;
}

```

```

    }
    if (crawledList.contains(link)) {
        continue;
    }
    linkList.add(link);
}
return (linkList);
}

private boolean searchStringMatches( String pageContents, String searchString,
boolean caseSensitive)
{
    String searchContents = pageContents;
    if (!caseSensitive) {
        searchContents = pageContents.toLowerCase();
    }
    Pattern p = Pattern.compile("[\\s]+");
    String[] terms = p.split(searchString);
    for (int i = 0; i < terms.length; i++) {
        if (caseSensitive) {
            if (searchContents.indexOf(terms[i]) == -1) {
                return false;
            }
        } else {

```

```

if (searchContents.indexOf(terms[i].toLowerCase()) == -1) {
return false;
}
}
}
return true;
}

public void crawl( String startUrl, int maxUrls, boolean limitHost, String
searchString, boolean caseSensitive)
{
HashSet crawledList = new HashSet();
LinkedHashSet toCrawlList = new LinkedHashSet();
toCrawlList.add(startUrl);
while (crawling && toCrawlList.size() > 0)
{
if (maxUrls != -1) {
if (crawledList.size() == maxUrls) {
break;
}
}

String url = (String) toCrawlList.iterator().next();
toCrawlList.remove(url);

URL verifiedUrl = verifyUrl(url);

```

```

if (!isRobotAllowed(verifiedUrl)) {
    continue;
}
updateStats(url, crawledList.size(), toCrawlList.size(), maxUrls);
crawledList.add(url);
String pageContents = downloadPage(verifiedUrl);
if (pageContents != null && pageContents.length() > 0)
{
    ArrayList links = retrieveLinks(verifiedUrl, pageContents, crawledList, limitHost);
    toCrawlList.addAll(links);
    if (searchStringMatches(pageContents, searchString, caseSensitive))
    {
        addMatch(url);
    }
}
updateStats(url, crawledList.size(), toCrawlList.size(), maxUrls);
}
}

public static void main(String[] args) {
    EbookCrawler crawler = new EbookCrawler();
    crawler.show();
}
}

```

4.2 Output

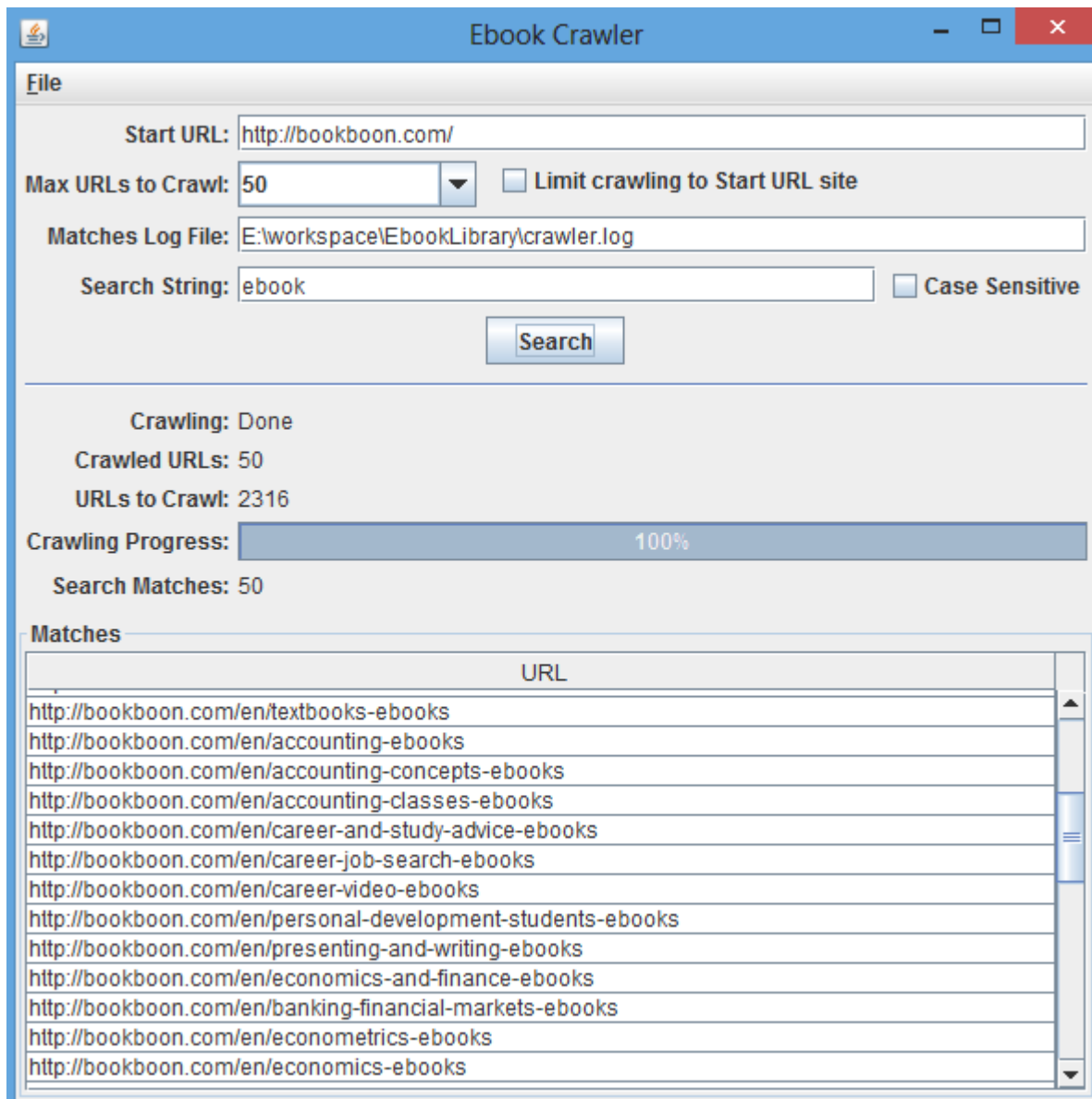
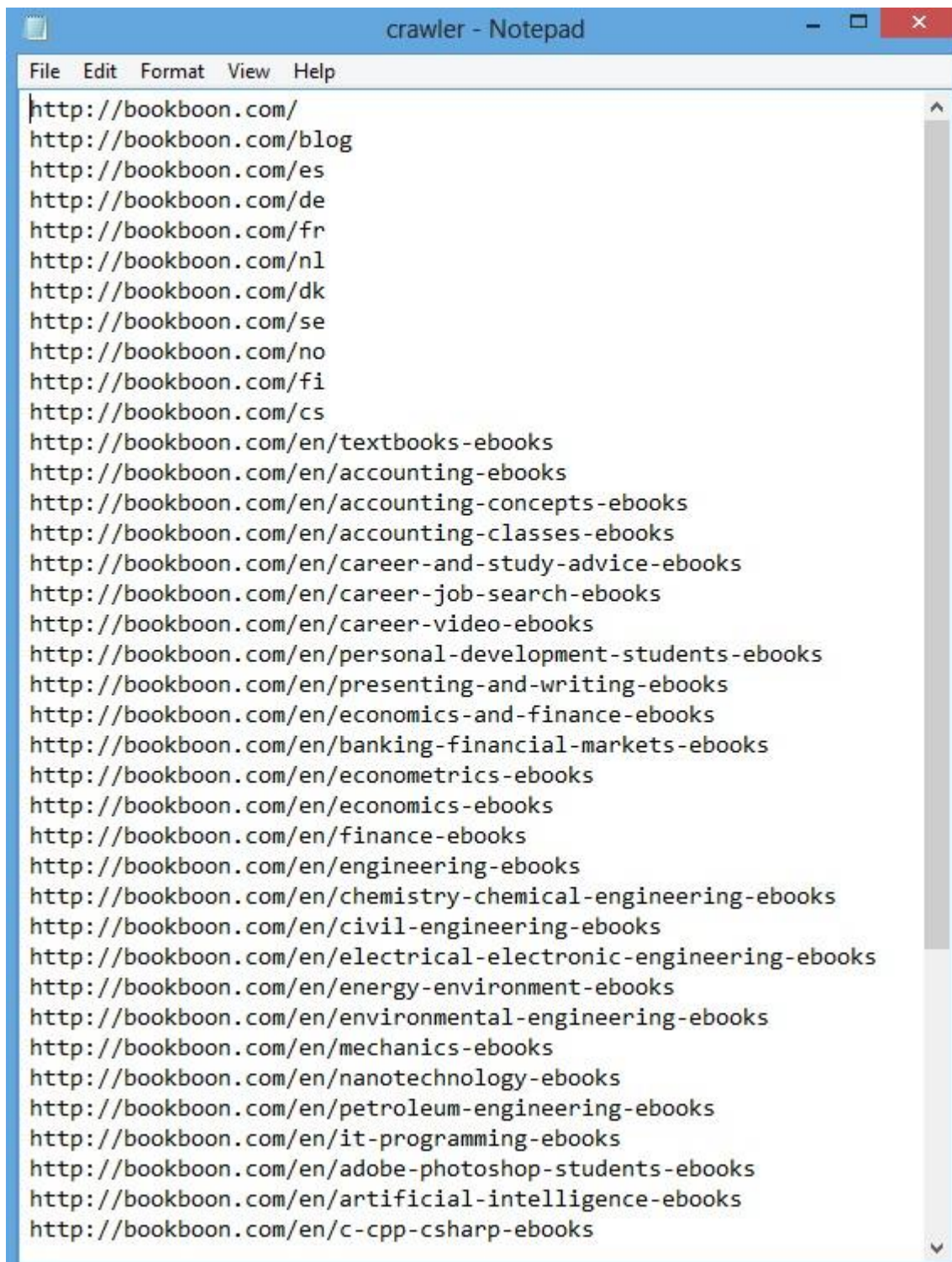


Figure 4.1 Output of eBook Crawler



```
File Edit Format View Help
http://bookboon.com/
http://bookboon.com/blog
http://bookboon.com/es
http://bookboon.com/de
http://bookboon.com/fr
http://bookboon.com/nl
http://bookboon.com/dk
http://bookboon.com/se
http://bookboon.com/no
http://bookboon.com/fi
http://bookboon.com/cs
http://bookboon.com/en/textbooks-ebooks
http://bookboon.com/en/accounting-ebooks
http://bookboon.com/en/accounting-concepts-ebooks
http://bookboon.com/en/accounting-classes-ebooks
http://bookboon.com/en/career-and-study-advice-ebooks
http://bookboon.com/en/career-job-search-ebooks
http://bookboon.com/en/career-video-ebooks
http://bookboon.com/en/personal-development-students-ebooks
http://bookboon.com/en/presenting-and-writing-ebooks
http://bookboon.com/en/economics-and-finance-ebooks
http://bookboon.com/en/banking-financial-markets-ebooks
http://bookboon.com/en/econometrics-ebooks
http://bookboon.com/en/economics-ebooks
http://bookboon.com/en/finance-ebooks
http://bookboon.com/en/engineering-ebooks
http://bookboon.com/en/chemistry-chemical-engineering-ebooks
http://bookboon.com/en/civil-engineering-ebooks
http://bookboon.com/en/electrical-electronic-engineering-ebooks
http://bookboon.com/en/energy-environment-ebooks
http://bookboon.com/en/environmental-engineering-ebooks
http://bookboon.com/en/mechanics-ebooks
http://bookboon.com/en/nanotechnology-ebooks
http://bookboon.com/en/petroleum-engineering-ebooks
http://bookboon.com/en/it-programming-ebooks
http://bookboon.com/en/adobe-photoshop-students-ebooks
http://bookboon.com/en/artificial-intelligence-ebooks
http://bookboon.com/en/c-cpp-csharp-ebooks
```

Figure 4.2 – Log File of Output

CONCLUSION:

I found this project quite interesting but time consuming. During implementation phase i faced many problems to build this project because in this project there are three phases crawling for eBook query , indexing the result and ranking the results. Due to time consuming modules I am able to complete only one phase of project which is crawling for the ebook.

So, the crawling phase of eBook Crawler is successfully implemented and it gives the relevant links of the eBook and store those results in a log file .

Bibliography

- [1] **Crawling the web: Discovery and Maintenance of large scale web data** by Junghoo Cho, Nov 2001
- [2] **Debajyoti Mukhopadhyay, Arup Biswas, Sukanta Sinha; *A New Approach to Design Domain Specific Ontology Based Web Crawler*; 10th International Conference on Information Technology, ICIT 2007 Proceedings; Bhubaneswar, India; IEEE Computer Society Press, California, USA; December 17-20, 2007; pp.289-291.**
- [3] **Jeff Allen;Indexing Web Crawler CSE 8337 - Margaret Dunham , 2009**
- [4] **The Robots Exclusion Protocol standard is described at <http://www.robotstxt.org/wc/exclusion.html>**
- [5] **Prioritization of Domain-Specific Web Information Extraction by Jian Huang and Cong Yu , 2010**
- [6] **A Distributed Approach to Crawl Domain Specific Hidden Web by Lovekesh kumar Desai , 2007**
- [7] **Indexing Web Crawler CSE 8337 – Margaret Dunham ,Spring 2009, Jeff Allen**
- [8] **Google : Crawling and Indexing at <https://www.google.co.in/insidesearch/howsearchworks/crawling-indexing.html>**
- [9] **Google: Page Rank at <https://support.google.com/webmasters/answer/70897?hl=en>**
- [10]**Wikipedia: Stemming at <http://en.wikipedia.org/wiki/Stemming>**
- [11]**Wikipedia: Focused Crawler at http://en.wikipedia.org/wiki/Focused_crawler**
- [12] **Wikipedia: Robots.txt at http://en.wikipedia.org/wiki/Robots_exclusion_standard**
- [13] **Scribd: Stemming Algorithm at <http://www.scribd.com/doc/81017602/36/Different-Stemming-Algorithms>**
- [14] **Crawling web with Java by Herbert Schildt and James Holmes, 2004**