CS104 PROJECT

PYTHON WEB CRAWLER

Sambhav Jain

June 2023

Contents

1 Introduction

A web crawler, also known as a web spider or web robot, is a software program that systematically navigates through the internet to discover and retrieve information from websites. It operates by following links from one web page to another and indexing the content it finds along the way.

The main purpose of a web crawler is to gather data from various websites and build an index or database that can be used for various purposes. Search engines like Google use web crawlers to collect information about web pages, analyze their content, and make the information searchable for users. Other applications of web crawlers include data mining, website scraping, website monitoring, and content validation.

2 Understanding The Code

2.1 Finding Unique Links and Analyzing Web page Elements

This task is done by the "find_unique_links" function. It
1.) Opens the webpage using the requests library, making an HTTP GET request to retrieve the webpage content.
2.) Measures the response time of the request.
3.) Decodes the HTML content of the page using UTF-8 encoding, addressing any potential decoding issues.
4.) Uses BeautifulSoup, a Python library for parsing HTML, to parse the HTML content and create a BeautifulSoup object called soup.
5.) Extracts the base domain from the provided URL using the urlparse function.
6.) Initializes sets to store unique links, internal links (links within the same domain), and external links (links outside the domain).
7.) Finds elements in the HTML with tags 'a','img', 'script', and 'link' using the find_all method of soup.
8.)Iterates over each element and retrieves the 'href' attribute and 'src' attribute.
9.) If an 'href' attribute exists, creates an absolute URL by combining it with the provided URL using urljoin. Adds the absolute URL to the set of unique links and checks if it belongs to the same domain. If it does, adds it to the set of internal links.
10.) If a 'src' attribute exists, follows the same steps as for 'href' to obtain the absolute URL, add it to the set of unique links, and determine if it belongs to the same domain or external domain.
11.) Returns the sets of unique links, internal links, and external links, along with the response time of the request.

HTML parsing explanation:
HTML parsing refers to the process of extracting structured data from HTML documents. In the context of this function, HTML parsing involves using the BeautifulSoup library to parse the HTML content of a webpage. It allows us to navigate and search for specific elements or attributes within the HTML document, extract information, and perform various tasks like finding links, images, scripts, and stylesheets on a webpage. By parsing the HTML, we can extract relevant information and analyze the structure of the webpage.

2.2 Recursive Web Crawling

The code utilizes two functions to handle different crawling scenarios. The "recursive_crawling" function is employed for crawling up to a specified threshold, while the "infinite_web_crawling" function enables infinite recursion.
1)The function takes the following parameters:

internal_links_passed: A set of internal links to crawl.
depth: The current depth of recursion.
depth_threshold: The maximum depth of recursion.
output_file: The file to write the output to. If None, the output is printed to the console.

2.)The categories dictionary maps file extensions to categories for better organization of files.
3.)The file_categories dictionary is initialized with a 'miscellaneous' category to store files with unknown or unsupported extensions.
4.)Empty sets are created for internal_links, all_links, and external_links.The function iterates over each link in the internal_links_passed set, calling the find_unique_links function to obtain the unique links, internal links, external links, and response time.
5.)The function categorizes the files based on their extensions. Each file link is split to extract the file extension, which is then used to determine the category using the categories dictionary. The file is added to the corresponding category set in the file_categories dictionary.
6.)The output is generated by appending relevant information to the output list. If include_internal is True, the internal links are added to the output. Similarly, if include_external is True, the external links are added. The recursion level and the total number of files found are also included in the output.
7.)The function iterates over the file categories and adds the category name and the number of files in that category to the output. It then appends each file link under the respective category.
8.)If output_file is None, the output is printed to the console. Otherwise, the output is written to the output_file.
9.)If the current depth is greater than 1 and there are internal links, the recursive_crawling function is recursively called with the updated internal_links set, reduced depth, and the same parameters.

2.3 Infinite Recursive Crawling

This part is similar to recursive web crawling except the point that in this function the recursion stops only when either there are no more internal links to crawl or the links to be crawled are already being crawled. Thus , there are two conditions to continue recursion
len(internal_links) == 0 or internal_links == internal_passed_links.

2.4 Roles of the Imported Libraries

The following libraries are used in the code:

- `argparse`: Allows the program to accept command-line arguments.
- `requests`: Sends HTTP requests and handles responses.
- `warnings`: Handles warning messages, including `InsecureRequestWarning`.
- `BeautifulSoup` (`bs4`): Parses HTML and XML documents.
- `urllib.parse`: Provides functions for working with URLs, including `urljoin` and `urlparse`.
- `time`: Works with time-related operations.
- `matplotlib.pyplot` (`plt`): Generates plots and visualizations.
- `os`: Performs file and directory operations.
- `matplotlib.backends.backend_pdf`: Supports creation of PDF files from matplotlib figures.
- `PyPDF2`: Works with PDF files, including reading and merging.

2.5 Command-Line Argument Parsing and Configuration

Argument parsing is a technique used to parse and process command-line arguments passed to a program. It allows users to provide input and customize the behavior of the program without modifying the source code. The argparse library in Python provides a convenient way to define, parse, and handle command-line arguments.

An ArgumentParser object named parser is created. It is initialized with a description that provides a brief explanation of what the program (web crawler) does. The ArgumentParser object is then used to define various command-line arguments that the program can accept.

3 Additional Features

3.1 Crawl Statistics

The `CrawlStatistics` class provides essential metrics and statistics for web crawling processes. It is designed to keep track of important information during crawling and evaluate the efficiency and performance of a web crawler. The following elements are included in the `CrawlStatistics` class:

- `start_time`: This attribute stores the starting time of the web crawling process. It helps measure the overall duration of the crawling operation.
- `pages_crawled`: This attribute keeps track of the total number of pages that have been successfully crawled. It helps assess the extent of the crawling process.
- `total_response_time`: This attribute accumulates the response times of all crawled pages. It aids in calculating the average response time.
- `error_count`: This attribute counts the number of errors encountered during the crawling process. It provides insights into potential issues or challenges faced by the crawler.

The `CrawlStatistics` class offers the following methods:

- `increment_pages_crawled()`: This method increments the count of crawled pages by one. It is called whenever a page is successfully crawled.
- `add_response_time(response_time)`: This method takes the response time of a crawled page as input and adds it to the total response time. It enables the calculation of the average response time.
- `increment_error_count()`: This method increments the error count by one. It is used when an error occurs during the crawling process.
- `get_average_response_time()`: This method calculates the average response time of the crawled pages by dividing the total response time by the number of crawled pages. It provides insights into the overall speed of page retrieval.
- `get_elapsed_time()`: This method calculates the elapsed time of the crawling process by subtracting the start time from the current time. It helps determine the duration of the crawling operation.
- `get_error_rate()`: This method calculates the error rate of the crawled pages by dividing the error count by the number of crawled pages and multiplying it by 100 to obtain a percentage. It allows for the assessment of the error-proneness of the crawled content.

```
(base) sambhavjain@Sambhavs-MacBook-Pro Downloads % python3 web.py -u https://www.iitb.ac.in/ -t 2 -o output.txt
Output file: output.txt


--- Crawl Statistics ---
Pages Crawled: 206
Average Response Time: 0.02 seconds
Elapsed Time: 12.82 seconds
Error Rate: 0.00%
(base) sambhavjain@Sambhavs-MacBook-Pro Downloads %
```

Figure 1: Crawl Statistics Usage

3.2 Generating Pie Charts for File Categories at Each Recursion Level

The `pie_chart` function is responsible for generating pie charts that represent the distribution of different file categories at each recursion level in the web crawling process. This function takes three parameters: `file_categories`, `pie_chart_file`, and `recursion_level`.

The `file_categories` parameter is a dictionary that contains the file categories as keys and sets of corresponding files as values. This information is used to determine the labels and sizes for the pie chart.

The `pie_chart_file` parameter specifies the file path where the generated pie charts will be saved as a PDF file.

The `recursion_level` parameter represents the current recursion level, which is used to add a descriptive title to each pie chart.

Inside the function, the labels and sizes are extracted from the `file_categories` dictionary to prepare the data for the pie chart. Then, a pie chart is created using the `pyplot.pie` function from the Matplotlib library. The `autopct` argument formats the percentage values displayed on the chart, and the `startangle` argument sets the initial rotation angle of the pie slices.

A title is set for the pie chart using the `ax.set_title` function, incorporating the `recursion_level` parameter.

Finally, the generated pie chart is saved to the specified `pie_chart_file` in PDF format. If the PDF file is empty, the chart is directly saved using the `PdfPages` context manager. Otherwise, a temporary PDF file is created to save the current chart, and then both the existing and new PDF files are merged using the `PdfMerger` class from the PyPDF2 library. The temporary PDF file is then removed.

The `pie_chart` function is useful in visually representing the distribution of file categories at different recursion levels in the web crawling process. It allows for easy analysis and identification of the types and proportions of files encountered during the crawl, helping in understanding the content structure of a website.

3.3 Inclusion of Internal and External Links in Web Crawler Output

When the -I option is specified, the web crawler will include the internal links in the output. Internal links refer to the links that belong to the same domain or website being crawled. These links are useful for understanding the structure and connectivity within the website.

Similarly, when the -e option is specified, the web crawler will include the external links in the output. External links are links that lead to different domains or websites. Including external links in the output can provide insights into the websites or resources that are referenced by the crawled website.

By using these options, the user can customize the output of the web crawler to include or exclude internal and external links based on their requirements. This feature enhances the flexibility and usefulness of the web crawler by providing additional information about the linked resources.

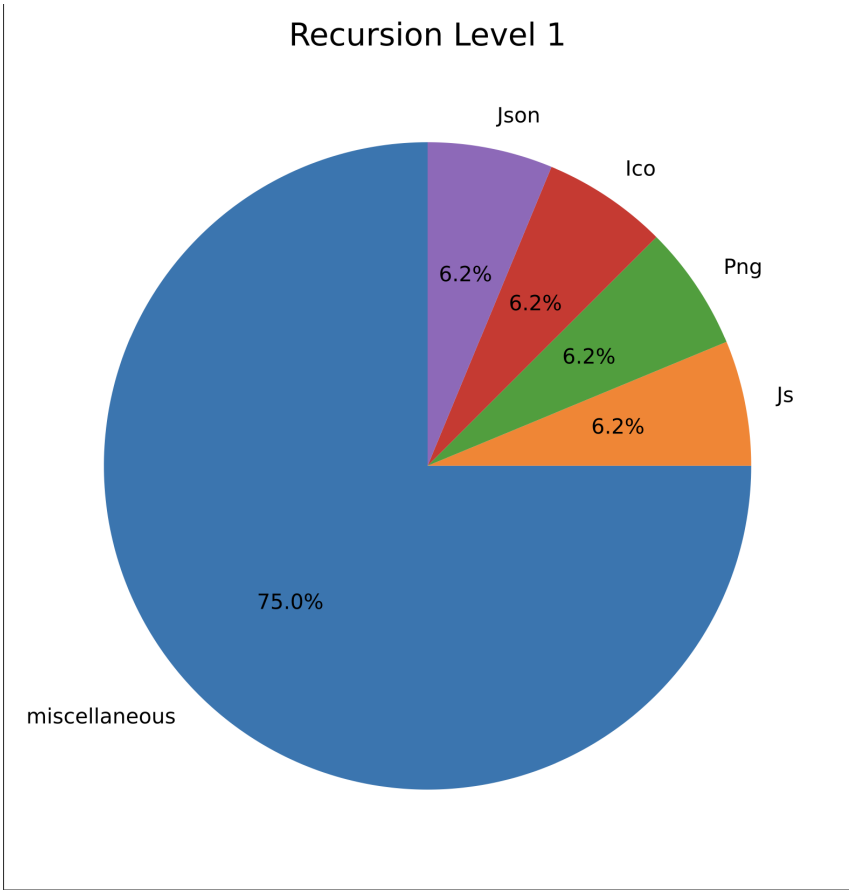

Figure 2: Pie Chart Generated

4 Usage

To run the web crawler, execute the following command:

```
python3 web_crawler.py [options]
```

5 Command Line Options

The following command line options are available for customizing the behavior of the web crawler:

- `-h, --help`: Show the help message and exit.
- `-u URL, --url URL`: Specify the URL to crawl. This option is necessarily required.
- `-t THRESHOLD, --threshold THRESHOLD`: Set the recursion threshold. This determines the depth of the crawling process. Only positive recursion values are valid.
- `-o OUTPUT, --output OUTPUT`: Specify the output file to save the results. If not provided, the results will be printed to the console.
- `-p, --plot`: Enable plotting of pie charts. This option visualizes the distribution of file types.This option requires -pc option to be used together.

- `-i, --include-internal`: Include internal links in the output. Internal links belong to the same domain as the crawled website.
- `-e, --include-external`: Include external links in the output. External links lead to different domains or websites.
- `-pc PIE_CHART, --pie-chart PIE_CHART`: Specify a PDF file to save the pie charts generated by the script.

6 Examples

Here are some examples of how to use the `web_crawler.py` script with different options:

- `python web_crawler.py -u http://example.com -t 3 -o output.txt`: Crawl the website `http://example.com` up to a depth of 3 and save the output to `output.txt`.
- `python web_crawler.py -u http://example.com -t 5 -o output.txt -p`: Crawl the website `http://example.com` up to a depth of 5, save the output to `output.txt`, and plot pie charts.
- `python web_crawler.py -u http://example.com -t 2 -o output.txt -i -e`: Crawl the website `http://example.com` up to a depth of 2, save the output to `output.txt`, and include both internal and external links in the output.

7 How I implemented it ?

The web crawler script is a collaborative effort, incorporating code from both external sources and my own modifications. I leveraged existing code for crucial tasks like HTML parsing using BeautifulSoup and segregating files into specific categories. These components were found online and subsequently tailored to my specific needs.

The core functionality of the script, involving recursive crawling and traversing web pages, was implemented by me. This enables the script to crawl multiple levels of web pages and gather information about links and file types. I also introduced additional features to enhance the functionality, such as crawl statistics to track the number of pages crawled, response times, and error counts.

The code for generating pie charts was adapted from an external source, and I further customized it to visualize the distribution of file types. By aggregating file categories and their respective counts, the script produces informative pie charts for each recursion level.

References

[1] Chatgpt: Large-scale language model for conversational ai. https://www.chatgpt.com, 2023. Accessed: June 10, 2023.

[2] Stack overflow. https://www.stackoverflow.com, 2023. Accessed: June 10, 2023.

[3] Topcoder. https://www.topcoder.com, 2023. Accessed: June 10, 2023.

[2] [1] [3]