CS104 Project Report: Web Crawler

Anirudh Garg
22B1005

June 15, 2023

1 Introduction

This code implements a web crawler, which is a program that automatically traverses through web pages and extracts information from them. The purpose of this specific web crawler is to extract various types of files and links from a given website. It starts by visiting the initial URL provided and then recursively crawls through internal links within a specified depth limit. The modifications made to the code have also been discussed, including the changes in functionality and structure. The modifications aim to improve the code's readability, maintainability, and extensibility. The modified code enhances the crawling process by adding features such as file downloading, file size calculation, and improved output formatting.

2 What it does ?

During the crawling process, the code parses the HTML content of each visited page using the BeautifulSoup library. It extracts links from ¡a¿ tags, file URLs from ¡link¿ and ¡script¿ tags, and image URLs from ¡img¿ tags. It categorizes these URLs based on their file extensions and stores them in a dictionary called output_format. Additionally, it can download specific files by their extensions if specified.
Firstly, it uses the requests library to send HTTP requests and retrieve the web page content. Then, it utilizes the BeautifulSoup library to parse the HTML content and extract relevant information.
The code maintains a set called visited_links to keep track of the visited links, ensuring that each link is processed only once and avoiding duplicate crawling. It also utilizes the urlparse library to parse and manipulate URLs.
To categorize the extracted URLs, the code employs the output_format dictionary. It categorizes the URLs into different file types, such as HTML, CSS, JavaScript, images (JPG, PNG, GIF), ICO files, and other file types. It further distinguishes external links that point to external websites.
The code provides the functionality to download specific files by their extensions. It saves the downloaded files in a folder named after the domain and the download extension.
The main function, main(), handles the command-line arguments using the argparse library. It initiates the crawling process, collects the extracted information, and generates the output.

3 Customisations

These customisations enhance the versatility and utility of the web crawler code, making it adaptable to different use cases and empowering users to extract and analyze website content efficiently.

3.1 -S or `--file_sizes`

This option calculates and displays the file sizes of each categorized file type. When this flag is enabled, the code retrieves the file sizes by sending additional HTTP requests to each file URL and calculates the size based on the response content length. It then includes the file sizes in the output, providing insights into the size distribution of different file types on the crawled website.
The -S option enables users to gain insights into the file size distribution on the crawled website. This information can be valuable for various purposes, such as optimizing website performance, identifying large files that may affect loading speed, or understanding the overall resource usage.

3.2 -D or --download_extension

This option allows users to specify a particular file extension to download from the website. When this argument is provided, the code identifies files with the specified extension and downloads them into a separate folder. This customization is helpful when there is a need to extract specific file types from a website for offline analysis or further processing.
The -D option facilitates the selective download of specific file types. It allows users to extract only the files they are interested in, saving time and storage space. This customization is particularly useful when conducting targeted analysis or collecting specific types of data from a website.

4 Comparing Final Code (Code 1) and Initial Code (Code 2)

4.1 Imports and Global Variables

Code 1:

```
import requests
import argparse
import os
from urllib.parse import urlparse, urljoin

visited_links = set()
output_format = {'jpg': set(), 'png': set(), 'gif': set(), 'ico': set(), 'other': set(),
    'external': set()}
links_to_crawl = set()
```

Code 2:

```
import requests
from bs4 import BeautifulSoup
import argparse
from urllib.parse import urlparse, urljoin

visited_links = set()
```

- In Code 1, an additional import statement `import os` is added, enhancing the functionality. - Code 1 introduces the `output_format` dictionary to categorize links based on their file extensions and source, making it more structured and useful. - Code 1 includes a `links_to_crawl` set to keep track of links to be crawled, providing better control over the crawling process.

4.2 get_links Function

[**2**]
Code 1:

```
def get_links(url, current_iteration, max_iterations, base_url, download_extension):
    # ...
```

Code 2:

```
def get_links(url, base_url):
    # ...
```

- In Code 1, the `get_links` function accepts additional parameters `current_iteration`, `max_iterations`, and `download_extension`, making it more versatile and configurable. - The parameters in Code 1 allow controlling the maximum number of iterations and the file extension to be downloaded, providing greater flexibility in the crawling process.

4.3 download_file Function

Code 1:

```
def download_file(url, base_url, download_extension):
    # ...
```

Code 2: (No equivalent function in Code 2)
- Code 1 introduces the `download_file` function, which handles the downloading of files from the specified URL. This crucial functionality is completely missing in Code 2, making it significantly less usable.

crawl Function

Code 1:

```
def crawl(base_url, max_iterations, download_extension, output_file):
    # ...
```

Code 2:

```
def get_links(url, base_url):
    # ...
```

- Code 1 includes the `crawl` function, which acts as the coordinator for the web crawling process, encompassing multiple functionalities. - The `crawl` function in Code 1 calls the `get_links` function recursively and performs file downloading and categorization, making it a comprehensive and usable solution. - Code 2 only includes the `get_links` function, which solely extracts and prints links, lacking the additional functionalities of file downloading and categorization. As a result, it cannot be considered a complete and usable solution.

4.4 Main Execution

Code 1:

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Web Crawler")
    parser.add_argument("base_url", help="Base URL to crawl")
    parser.add_argument("--max_iterations", type=int, default=10, help="Maximum number
    of iterations")
    parser.add_argument("--download_extension", help="File extension to download")
    parser.add_argument("--output_file", help="Output file name")
    args = parser.parse_args()

    crawl(args.base_url, args.max_iterations, args.download_extension, args.output_file)
```

Code 2:

```
if __name__ == "__main__":
    base_url = input("Enter base URL to crawl: ")

    get_links(base_url, base_url)
```

- Code 1 includes a well-structured argument parsing using the `argparse` module, allowing customization of various crawling parameters. - Code 1 provides the `--download_extension` and `--output_file` arguments, enabling the user to specify the file extension to download and the output file name for storing categorized links, respectively. - Code 2 lacks these additional arguments, limiting its functionality and usability.

Overall Comparison

Code 1 has been heavily edited and enhanced to address the limitations of Code 2, making it a superior solution. Here are the key advantages of Code 1:

1. Code 1 includes additional functionalities such as file downloading and categorization, which are completely absent in Code 2.
2. Code 1 introduces the `output_format` dictionary to categorize links, providing a more structured representation of crawled links.
3. Code 1 allows customization of the maximum number of iterations through the `max_iterations` argument, providing better control over the depth of the web crawl.
4. Code 1 includes the `--output_file` argument to specify an output file name for storing the categorized links, enhancing usability and convenience.
5. Code 1 includes error handling to handle exceptions during the crawling process, ensuring better robustness.

In summary, Code 1 is significantly more usable, versatile, and superior due to its additional functionalities, flexibility, and error handling capabilities. On the other hand, Code 2 is incomplete and lacks essential features, making it unusable for practical web crawling tasks.

Running the Web Crawler Code

1. Install Dependencies

- Make sure you have Python installed on your system.
- Install the required Python libraries by running the following command in your terminal:

```
pip install requests beautifulsoup4 argparse
```

2. Save the Code

- Copy the complete code provided earlier into a file and save it with a ".py" extension (e.g., "web_crawler.py").

3. Basic Usage

- To run the web crawler without any customizations, use the following command on the terminal with the directory in which you have the `"web_crwaler.py"` file:

```
python web_crawler.py -u <website_url> -t <recursion_threshold> -o <output_file>
```

Replace `<website_url>` with the URL of the website you want to crawl and `<recursion_threshold>` with the desired depth limit for crawling. For example:

```
python web_crawler.py -u https://example.com -t 3 -o output.txt
```

4. Optional Customisations

- To enable the file size calculation and display in the output, use the `-S` or `--file_sizes` option:

```
python web_crawler.py -u <website_url> -t <recursion_threshold> -S
```

- To download files of a specific extension, use the `-D` or `--download_extension` option:

```
python web_crawler.py -u <website_url> -t <recursion_threshold> -D <file_extension>
```

Replace `<file_extension>` with the desired file extension (e.g., "pdf", "jpg", "css"). Only files with this extension will be downloaded. They will be saved in a folder in the format of `<url>.<file_extension>`.

5. Output

- By default, the output will be displayed in the console.
- To save the output to a file, use the `-o` or `--output` option followed by the desired file name:

```
python web_crawler.py -u <website_url> -t <recursion_threshold> -o <output_file_name>
```

Replace `<output_file_name>` with the desired file name (e.g., "output.txt"). The output will be saved in the specified file.

6. Review the Output

- After running the command, the web crawler will crawl through the website and display or save the extracted information based on the provided options.
- The output will include the categorized URLs, external links, and, if enabled, the file sizes of each file type.

Now you can run the web crawler code and explore different websites, customize the crawling behavior, and extract specific file types based on your requirements.