Python Web Crawler Report

Saranya Darisipudi

June 14, 2023

Contents

1 Description

This report presents the results of a web crawling operation performed using the provided code. The purpose of the web crawler is to extract and analyze links from a given URL up to a specified depth of recursion. The extracted links are categorized based on their type and stored in separate lists. The report includes detailed information about the extracted links and provides visualizations to illustrate the distribution of different file types.

2 Code

```
# Importing necessary libraries
import requests
from bs4 import BeautifulSoup
import argparse
import urllib.parse
import urllib3
import matplotlib.pyplot as plt

# Disable SSL/TLS warnings to suppress InsecureRequestWarning
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

# Set up command-line argument parser
parser = argparse.ArgumentParser(description='Web Crawler')
parser.add_argument('-u', '--url', help='Input url')
parser.add_argument('-t', '--level', help='threshold of recursion')
parser.add_argument('-o', '--output_file', help='Output file')

# Parse command-line arguments
args = parser.parse_args()

# Initializing required lists
total_links = []  # Nested list - For storing all links
internal_links = []  # Nested list - For storing all internal links
external_links = []  # Nested list - For storing all external links
types = []  # Nested list - For storing the types of all links

# Assigning variables to the input parameters
i_dep = int(args.level)
i_url = args.url
o_url = args.output_file

# Opening output file
output_file = open(o_url, "w")

# Check if depth is zero
if i_dep == 0:
    output_file.write(f"Error: Depth should be greater than zero.")
    exit(1)

# To get base domain of input url
base_domain = urllib.parse.urlparse(i_url).netloc


# Function for finding the extension of url, used for segregating files into different
    types
def type_of_link(url):
    # Parsing the URL
    parsed_url = urllib.parse.urlparse(url)
    # To get the path from the parsed URL
    path = parsed_url.path
    # Split the path into parts
```

```
    parts = path.split('/')
    # To get the last part (filename)
    filename = parts[-1]
    # Split the filename to get extension
    if '.' in filename:
        extension = filename.split(".")[-1]
        return extension
    else:
        # For Miscellaneous files
        return "Misc"


# Recursive crawling function
def crawl(url, depth):
    """
        Recursive function to crawl the web pages and extract links.

        Args:
            url (str): The URL to crawl.
            depth (int): The current depth of recursion.

        Returns:
            None
    """
    # Base case: Stop crawling if depth is 0
    if depth > 0:
        try:
            # Send a GET request to the URL
            response = requests.get(url, verify=False)

            # Parse the HTML content using BeautifulSoup
            soup = BeautifulSoup(response.text, 'html.parser')

            # for listing all elements in particular recursion level
            # Determine the current recursion level, here, level will be equal to
                current recursion level - 1
            # Here, i_dep is the input depth
            level = i_dep - depth

            # Initialize lists for current recursion level A new empty list is appended
                to the following lists if
            # their length is less than or equal to current recursion level
            if len(total_links) <= level:
                total_links.append([])
            if len(internal_links) <= level:
                internal_links.append([])
            if len(external_links) <= level:
                external_links.append([])
            if len(types) <= level:
                types.append([])

            # Process the page (e.g., extract information, save data, etc.)
            # Find all the links on the page
            links = soup.find_all(['a', 'img', 'script', 'link'])

            # Here, depth is reduced by one for next recursion
            depth = depth - 1

            # Fill the lists with corresponding links
            for a_link in links:
                # Extract the URL from the 'href', 'src' attributes
                if a_link.has_attr('href'):
```

```
                    next_url = a_link.get('href')
                elif a_link.has_attr('src'):
                    next_url = a_link.get('src')
                else:
                    continue

                # Construct the absolute URL by joining the base URL and the relative URL
                next_url = urllib.parse.urljoin(url, next_url)
                # Append the next_url only if it is not already present in total_links
                    at this recursion level
                if next_url not in total_links[level]:
                    total_links[level].append(next_url)

                    # Determine the extension of next_url
                    extsn = type_of_link(next_url)

                    # Append the link's type only if it is not present already in types
                        at this recursion level
                    if extsn not in [item[0] for item in types[level]]:
                        types[level].append([extsn, []])

                    # Check if the link is internal or external
                    next_domain = urllib.parse.urlparse(next_url).netloc
                    next_scheme = urllib.parse.urlparse(next_url).scheme

                    if next_url and (next_scheme == "http" or next_scheme == "https")
                        and (
                            next_domain == base_domain or next_domain == ""):
                        # Recursively crawl the next URL with reduced depth
                        internal_links[level].append(next_url)
                        crawl(next_url, depth)
                    else:
                        external_links[level].append(next_url)

        except requests.exceptions.RequestException as er:
            print('Error:', er)


crawl(i_url, i_dep)

# Segregate the total_links based on their extension at each recursion level
for i in range(len(total_links)):
    for link_i in total_links[i]:
        ext_i = type_of_link(link_i)
        type_index = [x[0] for x in types[i]].index(ext_i)
        types[i][type_index][1].append(link_i)

# For output
a = len(total_links)

# For plotting the file types and their counts
rows = a // 2 + a % 2
cols = 2
fig, axs = plt.subplots(rows, cols, figsize=(8, 4 * rows), squeeze=False)
fig.suptitle('File␣Type␣Distribution␣for␣each␣Recursion␣level')

for b in range(a):
    c = b + 1
    total = len(total_links[b])
    internal = len(internal_links[b])
    external = len(external_links[b])
    output_file.write(f"At␣recursion␣level:␣{str(c)}\n")
```

```
    output_file.write(f"Total␣files␣found:␣{str(total)}\n\n")
    output_file.write("Segregating␣them␣into␣different␣types:\n")
    for type_info in types[b]:
        ext = type_info[0]
        link_list = type_info[1]
        ll = len(link_list)
        if ext == "Misc":
            output_file.write(f"\nMiscellaneous␣files:␣{str(ll)}\n")
            for link in link_list:
                output_file.write(f"{link}\n")
        else:
            output_file.write(f"\nFiles␣with␣extension␣'{ext}':␣{str(ll)}\n")
            for link in link_list:
                output_file.write(f"{link}\n")
    output_file.write(f"\n")
    output_file.write(f"Internal␣files␣found:␣{str(internal)}\n")
    for y in range(internal):
        output_file.write(f"{internal_links[b][y]}\n")
    output_file.write(f"\nExternal␣files␣found:␣{str(external)}\n")
    for z in range(external):
        output_file.write(f"{external_links[b][z]}\n")
    output_file.write(f"\n")
    labels = []
    values = []
    for element in types[b]:
        file_type = element[0]
        file_count = len(element[1])
        # Prepare data for the horizontal bar chart
        labels.append(file_type)
        values.append(file_count)
    # Plot the horizontal bar chart
    ax = axs[b // cols, b % cols]
    ax.barh(labels, values, color='orange', edgecolor='blue', height=0.7)
    ax.invert_yaxis()
    ax.set_xlabel('Count')
    ax.set_ylabel('File␣Type')
    ax.set_title(f'Recursion␣Level␣{b␣+␣1}')

# If the number of recursion levels is odd, remove the last subplot to maintain a
    balanced layout
if a % 2 == 1:
    fig.delaxes(axs[-1, -1])

plt.tight_layout()

# Save the output file
output_file.close()

# Display the plots
plt.show()
```

3 Implementation

The implementation of the Web Crawler is done using Python.

3.1 Libraries Used

The following libraries are imported for implementing the web crawler: `requests`, `BeautifulSoup`, `argparse`, `urllib.parse`, `urllib3`, and `matplotlib.pyplot`.

3.2 Command-Line Arguments

The `argparse` library is used to handle the command-line arguments for the web crawler. The following arguments are defined:

- `-u`: Specifies the input URL to start crawling from.
- `-t`: Specifies the depth of recursion, i.e., how many levels of links to follow.
- `-o`: Specifies the output file to save the results.

The command-line arguments are parsed, and their values are assigned to corresponding variables in the code. Also, several nested lists are initialized to store the corresponding files and their types at each recursion level: `total_links`, `internal_links`, `external_links`, `types`.

First, Depth of recursion is checked if it is not equal to zero or not. If it is equal to zero, it exits the code.

3.3 Functions

3.3.1 `type_of_link`

This function returns the extension of url, which is given as an argument to it. It first parses the url, and splits its path into parts through '/'. Then, it separates the last part from those, into further subparts through '.'. Then, it returns the last subpart, which is the extension of url. If there is no '.' in the last part, it just returns 'Misc' indicating that the given url is Miscellaneous.

3.3.2 `crawl`

The crawling process is performed recursively using the `crawl` function. This function takes a URL and the current depth of recursion as input. Here is an overview of the steps involved in the crawling process:

1. A GET request is sent to the specified URL using the `requests` library.
2. The HTML content of the web page is parsed using `BeautifulSoup`.
3. Next, a variable "level" is defined, as a measure of level of recursion. It is defined as i_dep - depth, where i_dep is the input depth, depth is the current depth of recursion. So, here, level will be equal to 0 for recursion level 1. This is useful while storing links, as at recursion level 1, they will be stored at index 0(level) in all lists.
4. Links are extracted from the page using `BeautifulSoup` methods.
5. The extracted links are categorized as internal or external based on the URL's domain.
6. The lists of links and their types are updated.
7. The `crawl` function is recursively called for internal links with reduced depth.

After the crawling process is complete, the code segregates the links based on their extension at each recursion level. It iterates over the lists of links and types, and writes the information to the output file. Finally, the code generates plots to illustrate the distribution of different file types at each recursion level. The plots are created using the `matplotlib` library.

4 Customization

- Total files at each recursion level are separated into internal and external files and their count is printed in output.
- Horizontal bar graphs will be plotted for count of files of a particular type at each recursion level, with X-axis as count of files, and Y-axis as types of files.

5 Compilation

Run the following command to execute the code:

```
python main.py -u <input-url> -t <depth> -o <output-file>
```

6 References

References

[1] URL: https://chat.openai.com/.

[2] URL: https://www.geeksforgeeks.org/matplotlib-axes-axes-bar-in-python/.

[3] URL: https://pythonguides.com/get-file-extension-from-file-name-in-python/.

I took the basic web crawler code from ChatGPT and then, modified accordingly, by initializing several lists, adding function for finding type of url, and segregating them accordingly, adding the print statements and finally included the code for plotting horizontal bar graphs.

7 Images

7.1 Output

These images show the final output.

```
At recursion level: 1
Total files found: 2

Segregating them into different types:

Files with extension 'js': 1
https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js

Miscellaneous files: 1
https://www.cse.iitb.ac.in/

Internal files found: 1
https://www.cse.iitb.ac.in/

External files found: 1
https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js

At recursion level: 2
Total files found: 44

Segregating them into different types:

Files with extension 'css': 5
https://www.cse.iitb.ac.in/css/materialize.min.css
https://www.cse.iitb.ac.in/css/material-icon.css
https://www.cse.iitb.ac.in/css/vega.css
https://www.cse.iitb.ac.in/user_css/_global.css
https://www.cse.iitb.ac.in/css/year_bar.css

Files with extension 'ico': 2
https://www.cse.iitb.ac.in/..images/favicon.ico
https://www.cse.iitb.ac.in/images/cse-ico.ico

Miscellaneous files: 7
https://fonts.googleapis.com/icon?family=Material+Icons
https://www.cse.iitb.ac.in/
https://www.cse.iitb.ac.in/about/
```

(a) Output 1

```
https://www.cse.iitb.ac.in/includes/jquery/_jquery_frontpage.js
https://www.cse.iitb.ac.in/js/materialize.min.js
https://www.cse.iitb.ac.in/js/vega.js
https://www.cse.iitb.ac.in/js/prefixfree.min.js
https://www.cse.iitb.ac.in/js/jquery-ui.min.js
https://www.cse.iitb.ac.in/includes/jquery/_jquery_sidebar.js

Files with extension 'php': 21
https://www.cse.iitb.ac.in/user_css/_sidebar.php
https://www.cse.iitb.ac.in/user_css/_main.php
https://www.cse.iitb.ac.in/user_css/_collapsible.php
https://www.cse.iitb.ac.in/user_css/_cards_standard.php
https://www.cse.iitb.ac.in/academics/programmes.php
https://www.cse.iitb.ac.in/academics/courses.php
https://www.cse.iitb.ac.in/academics/time-table.php
https://www.cse.iitb.ac.in/admission/pg.php
https://www.cse.iitb.ac.in/admission/btech.php
https://www.cse.iitb.ac.in/research/research.php
https://www.cse.iitb.ac.in/research/labs.php
https://www.cse.iitb.ac.in/people/faculty.php
https://www.cse.iitb.ac.in/people/students.php
https://www.cse.iitb.ac.in/people/others.php
https://www.cse.iitb.ac.in/about/news.php
https://www.cse.iitb.ac.in/research/talks.php
https://www.cse.iitb.ac.in/engage/join.php
https://www.cse.iitb.ac.in/engage/involve.php
https://www.cse.iitb.ac.in/about/reach.php
https://www.cse.iitb.ac.in/internal/calendar.php
https://www.cse.iitb.ac.in/index.php

Files with extension 'png': 1
https://www.cse.iitb.ac.in/images/iitblogowhite.png

Internal files found: 42
https://www.cse.iitb.ac.in/css/materialize.min.css
https://www.cse.iitb.ac.in/css/material-icon.css
https://www.cse.iitb.ac.in/css/vega.css
```

(b) Output 2

7.2 Plots

This image shows the plots generated after running the code.

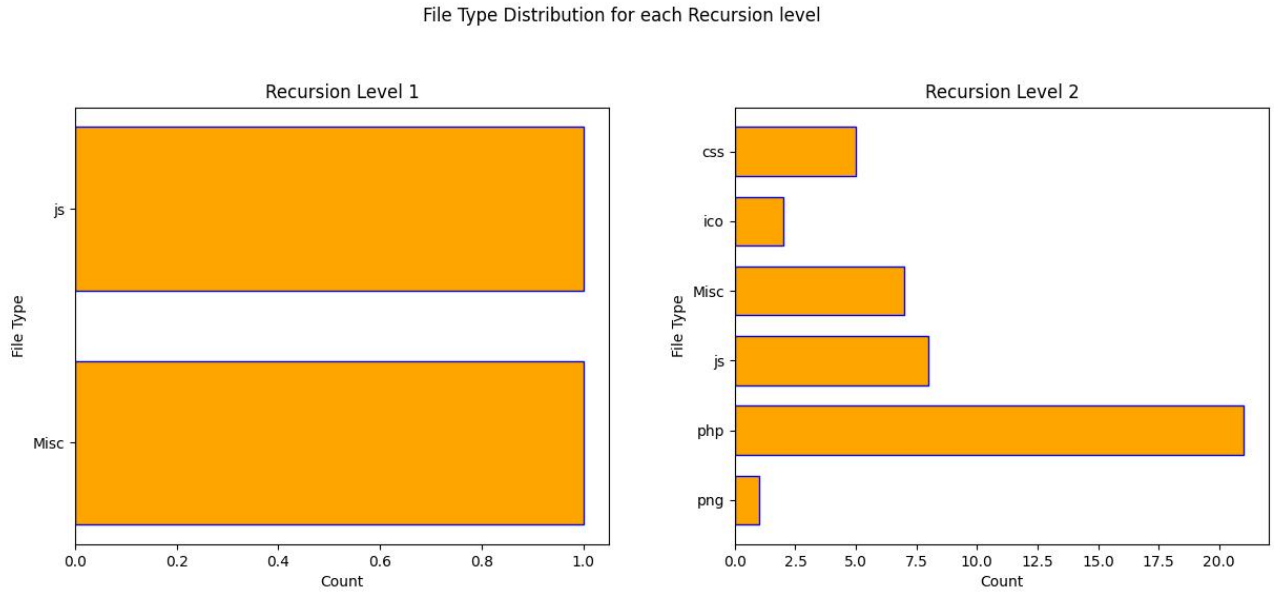

Figure 2: Plots