Crawler Design for Place data

Introduction

Crawler is a program used to fetch and download pages from the web. It fetches pages and extracts URLs for further crawling. They are majorly used in search engines. This report delves into a naïve implementation of a crawler using Python script with a simple content prioritization technique.

Methodology

The program uses the WikipediaAPI library as a source for the crawler to search for a user-inputted topic on Wikipedia. It then fetches the relevant Wikipedia pages based on the input. It utilizes PyQuery for parsing the webpage content. The core methodology includes extracting child URLs from the parsed webpage and shortlisting the pages to be displayed by counting occurrences of specific keywords in each child URL. These keyword counts are essential for prioritizing URLs.

API and Libraries used are as follows

1. WikipediaAPI: WikipediaAPI is a Python library that provides an easy-to-use interface for interacting with Wikipedia. It allows you to access Wikipedia content, search for articles, retrieve page summaries, and more. In the script, it's utilized for searching Wikipedia based on user input, fetching specific Wikipedia pages, and checking for page existence.
2. PyQuery: PyQuery is a Python library that allows parsing of HTML and XML documents. It simplifies the process of extracting data from web page. In the program PyQuery is used for parsing the content of webpages obtained from URLs, enabling extraction of specific elements and information.
3. urllib.parse: The urllib.parse module in Python provides functions for working with URLs. It includes various methods for URL encoding, decoding, joining base URLs with relative URLs, and other URL related manipulations.In the program it is utilized in conjunction with urljoin() function to create absolute URLs by resolving relative URLs against base URLs. This is essential for ensuring accurate and complete URLs for further processing.

Program Overview

The code comprises functions to search and crawl Wikipedia pages based on user input. The **extract\_and\_store\_child\_links** function extracts child URLs and counts occurrences of specific keywords. The code maintains a queue system, where URLs and their keyword counts are prioritized dynamically.

**Function descriptions are as follows**

* **fetch\_and\_parse\_url(url):**
  + **Input:** URL to fetch and parse.
  + **Output:** PyQuery object representing the parsed webpage.
  + **Description:** Uses PyQuery to fetch and parse the content of the given URL. Handles exceptions and returns **None** in case of errors.
* **extract\_and\_store\_child\_links(base\_url, pq\_doc):**
  + **Input:** Base URL and PyQuery object.
  + **Output:** List of tuples containing child URLs and their keyword counts.
  + **Description:** Extracts child URLs from the webpage and counts occurrences of the last part of the base URL in each child URL. Utilizes PyQuery for HTML parsing and urllib.parse to create absolute URLs.
* **count\_keyword\_occurrences(url, keyword):**
  + **Input:** URL and keyword to count occurrences.
  + **Output:** Number of occurrences of the keyword in the URL.
  + **Description:** Counts occurrences of the specified keyword in the given URL. Case-insensitive matching ensures accurate counts.
* **find\_nearest\_wikipedia\_page(keyword):**
  + **Input:** Keyword for which the nearest Wikipedia page is found.
  + **Output:** URL of the nearest Wikipedia page.
  + **Description:** Utilizes WikipediaAPI to find the nearest Wikipedia page URL for the entered keyword. Handles non-existent pages gracefully.
* **Queue System :**
  + The code maintains a queue, prioritizing URLs based on their keyword counts. URLs with higher keyword counts are given priority for further crawling.

Program

import wikipediaapi

from pyquery import PyQuery as pq

from urllib.parse import urljoin

wiki\_wiki = wikipediaapi.Wikipedia(

language='en',

user\_agent='YourAppName/1.0'

)

wiki\_wiki.user\_agent = 'YourAppName/1.0'

def fetch\_and\_parse\_url(url):

try:

response = pq(url=url)

return response

except Exception as e:

print(f"An error occurred while fetching {url}: {e}")

return None

def extract\_and\_store\_child\_links(base\_url, pq\_doc):

links = []

keyword = base\_url.split('/')[-1]

for link in pq\_doc('a[href]'):

href = pq(link).attr('href')

if href and not href.startswith('#'):

absolute\_url = urljoin(base\_url, href)

if absolute\_url not in links:

links.append((absolute\_url, count\_keyword\_occurrences(absolute\_url, keyword)))

return links

def count\_keyword\_occurrences(url, keyword):

return url.lower().count(keyword.lower())

def find\_nearest\_wikipedia\_page(keyword):

seed\_page = wiki\_wiki.page(keyword)

if seed\_page.exists():

return seed\_page.fullurl

else:

print(f"No Wikipedia page found for '{keyword}'")

return None

title = input("Enter a Wikipedia title: ")

seed\_url = find\_nearest\_wikipedia\_page(title)

if seed\_url:

print(f"Seed URL based on '{title}': {seed\_url}")

visited\_urls = []

queue = [(seed\_url, 0)]

max\_child\_urls\_to\_crawl = 10 # can be customized

while queue and len(visited\_urls) < max\_child\_urls\_to\_crawl:

url, keyword\_count = max(queue, key=lambda x: x[1])

queue.remove((url, keyword\_count))

pq\_doc = fetch\_and\_parse\_url(url)

if pq\_doc:

visited\_urls.append((url, keyword\_count))

child\_links = extract\_and\_store\_child\_links(url, pq\_doc)

for child\_url, child\_keyword\_count in child\_links:

queue.append((child\_url, child\_keyword\_count))

print("Crawling completed. Visited Child URLs are:")

for url, keyword\_count in visited\_urls:

print(f"URL: {url}")

Conclusion:

This Python crawler, utilizing WikipediaAPI and PyQuery, exemplifies streamlined web data extraction. While its current focus lies within Wikipedia, there's potential for expansion to diverse online platforms. Future enhancements could involve integrating advanced similarity measures for prioritization