HW#3 - Web Archiving

Ethan Landers

Due: October 20th by 11:59 PM EST

Methodology

Data Sources

The URIs in this assignment originate from Homework 2 for CS 532, where I created a URI mapping file named uri_mapping.txt. This file contains a dictionary that pairs the generated hash file names, which hold raw HTML content, with their corresponding URIs.

Tools Used

I utilized MemGator to complete this task. For each URI, I queried MemGator within the query_memgator() function in download_timemaps.py using:

```
~/MemGator/memgator -c 'ODU CS532 eland007@odu.edu' -a ~/MemGator/docs/
   archives.json -f JSON {uri}
```

Python Modules & Libraries Used

- import subprocess
  - This module assists with running external commands and interacting with the system shell.
- import time
  - This module assists with adding delays for pausing between requests.
- import os
  - This module assists with interacting with the file system.
- import json
  - This module assists with parsing and working with JSON data formats.
- from urllib.parse import urlparse
  - This function helps with parsing URLs into components.

- from prettytable import PrettyTable
  - This library helps with generating formatted tables in a readable format.
- from collections import defaultdict
  - This class is a dictionary-like container that provides values for non-existing keys.

Challenges and Considerations

Performance Issues

Gathering all the TimeMaps with MemGator through download_timemaps.py took a considerable amount of time, probably roughly over 8 hours. There were over 500 URIs to analyze to gather all the necessary TimeMaps, and MemGator contacts many archives to find relevant mementos, therefore taking a considerable amount of time.

Q1

There are three main files for this assignment: download_timemaps.py, analyze_mementos.py, and utils.py. For Q1, I used download_timemaps.py, which defines two functions: query_memgator() and download_timemap(). The former queries MemGator for a TimeMap of a given URI and saves the result, or writes a blank file if no memento exists. The latter drives the TimeMap download process for all relevant URIs.

In utils.py, the function load_uri_mapping() is used by both download_timemaps.py and analyze_mementos.py() to access uri_mapping.txt, generated in Homework 2. This file contains URIs and their associated hashes, providing the necessary data for the assignment.

```
def query_memgator(uri, output_file):

    command = f"~/MemGator/memgator -c 'ODU CS532 eland007@odu.edu' -a
    ~/MemGator/docs/archives.json -f JSON {uri}"

    try:
        result = subprocess.run(command, shell=True, capture_output=
    True, text=True, timeout=180)

        if result.returncode != 0:
            print(f"Error querying MemGator for {uri}: {result.stderr}"
    )
        else:
```

```python
            if result.stdout:
                print(f"TimeMap for {uri}: {result.stdout}")
            else:
                print(f"No TimeMap for {uri}.")

            with open(output_file, 'w') as f:
                f.write(result.stdout)

    except subprocess.TimeoutExpired:
        print(f"Query for {uri} timed out after 180 seconds.")

    time.sleep(10)

def download_timemap(uri_mapping_file, output_dir):

    os.makedirs(output_dir, exist_ok=True)

    uri_hash_map = load_uri_mapping(uri_mapping_file)

    uri_count = 1

    for hash_file, uri in uri_hash_map.items():
        output_file = os.path.join(output_dir, f"{hash_file}.json")
        query_memgator(uri, output_file)
        uri_count += 1

download_timemap("homework-2/uri_mapping.txt", "homework-3/timemaps")
```

Listing 1: download_timemaps.py

Q2

To answer Q2, I performed four tasks: handling TimeMaps, counting mementos, processing domains, and displaying tables.

First, the function load_timemap() loads TimeMaps, and analyze_timemaps() counts mementos across all TimeMaps in a directory.

For memento counting, count_mementos() counts mementos in a TimeMap, while count_memento_occurrences() tallies how often each memento appears, contributing to Table 1. find_top_mementos() identifies URIs with the most mementos for Table 2.

For domain analysis, get_core_domain() extracts the core domain from a URI. count_core_domain_frequencies() tracks domain frequencies based on memento counts, and find_most_frequent_domains() identifies the most frequent domains based on memento counts.

Finally, running analyze_mementos.py generates three tables that display the memento analysis:

Table 1: Distribution of Mementos Across URI-Rs

Mementos	**URI-Rs**
0	298
3	232

Table 2: Top URI-Rs With the Most Mementos”

URI-Rs With The Most Mementos	**Memento Count**
https://www.odu.edu/sci	3
https://arxiv.org/abs/1905.12607	3
https://arxiv.org/abs/1905.03836	3
https://www.odu.edu/partnerships/community	3
https://www.odu.edu/life/sports-recreation	3

Table 3: Top Domains With the Most Mementos

Domains With The Most Mementos	**Memento Count**
www.odu.edu/sci	102
news-un-org.translate.goog	96
dblp.uni-trier.de	84
arxiv.org	57
www.daad-ukraine.org	45

Q: What URI-Rs had the most mementos? Did that surprise you?

I was not able to obtain an accurate analysis at the URI-R level because out of the 530 analyzed URI-Rs, 232 of them had three mementos. Table 2 displays five URIs with a memento count of three. However, these URIs may represent the first ones encountered in the dataset with that memento count, which could lead to inaccuracies in analysis. Consequently, I decided to look at the domain level to determine which domains had the mementos created for them.

The URI-Rs that were gathered from Homework 1 that have the most mementos belong to the domain `www.odu.edu`. I’m not surprised by this as Old Dominion University is a large, reputable public university known for its research. Additionally, it has a .edu top-level domain, which is often associated with educational institutions.

References

- Defaultdict in Python, `https://www.geeksforgeeks.org/defaultdict-in-python/`
- JSON Python Library, `https://docs.python.org/3/library/json.html`
- MemGator Documentation, `https://github.com/oduwsdl/MemGator`
- Pretty Table Python Library, `https://pypi.org/project/prettytable/`
- Python Urllib Module, `https://www.geeksforgeeks.org/python-urllib-module/`
- Subprocess Python Library, `https://docs.python.org/3/library/subprocess.html`
- time.sleep() in Python, `https://www.geeksforgeeks.org/sleep-in-python/`