“Ctenidae” Web Crawler

Design Documentation

Design Work and Programming Completed By:

Caleb G. Lally

Table of Contents:

I. Abstract

The modern internet is a networked system of billions of devices communicating amongst each other as either clients or servers. Most internet activity consists of users both viewing webpages, as well as clicking links to other webpages. The rise of the search engine – applications designed to search through billions of webpages by parsing user queries to obtain desired information – was made possible by smaller component applications known as "web crawlers." The web crawler's function is simple from a design standpoint: scan webpages via a URL to find links to other webpages. By storing this information, a complete picture of the pages and resources offered by a website can be obtained. In order to maximize the efficiency of the application as well as to reduce load on the web server being prompted, competently constructed web crawlers will avoid redundantly visiting previously scanned web pages. It is necessary, therefore, to keep track of all visited pages in a data structure and search it for entries before opening new pages. The underlying goal of this project is to implement a web crawler that will simulate visiting live web pages by parsing a .json file, and utilize multithreading to concurrently execute its core algorithm.

II. Project Requirements

Requirements:

- Develop a web crawler to parse a .json file for additional links to visit.
- Store visited links in a data structure.
- Print report at the end of execution that lists:
    - Pages crawled successfully.
    - Duplicate pages skipped.
    - Invalid addresses skipped.

Constraints:

- The source code must be developed in either Java, Golang, or C#.
- Application must visit each valid page exactly once.
- Multithreading must be utilized to concurrently execute crawling algorithm.
    - Data protection must be used on data accessibly by other threads.

III. High-Level Design Summary

The Web Crawler solution consists of one Main class, and four auxiliary classes:

webCrawl:

- Class Definition
    - String jsonPage – used to hold the JSON to be parsed as a string
    - Pages testPage – constructed object to hold the object parsed from JSON
    - Thread crawlThread – used to implement concurrency
- Associated Methods
    - void run – used to run startCrawl method as a thread
    - void startCrawl – used to call the crawlLink and printReport methods
    - void crawlLink – primary processing method, used to parse the deserialized json
    - boolean searchForAddress – used for checking presence of given address in Pages
    - int getPage – used to find the associated element of list of addresses in Pages
    - webcrawl – default constructor
- Extensions and Libraries
    - Gson
    - crawlReportClass class

- Runnable interface

Page:

- Class Definition
  - public String address – used for deserializing addresses from json as string
  - public List<string> links – used for deserializing list of addresses from page
- Associated Methods
  - public Page withAddress – getter, returns address of Page object
  - public Page withLinks – getter, returns list of links for Page object
- Extensions and Libraries
  - Gson
  - List

Pages:

- Class Definition
  - public List<Page> pages – used for deserializing links from json as string list
- Associated Methods
  - public Pages withPages – getter, returns list of strings under pages object
- Extensions and Libraries
  - Gson
  - List
  - Page class

crawlReportClass:

- Class Definition
    - Hashtable successReport – used to store addresses of successful crawls
    - Hashtable duplicateReport – used to store addresses of skipped pages
    - Hashtable noPageReport – used to store links with no corresponding address
    - Hashtable visitedPages – used to track visited pages in tandem with other tables
    - int noPageCount – used to provide key for the noPageReport table
    - int successCount – used to provide key for the successReport table
    - int duplicateCount – used to provide key for the duplicateReport table
    - int visitedCount – used to provide key for the visitedPages table
- Associated Methods
    - public void duplicateReport – stores all addresses skipped
    - public void noPageReport – stores all links with no corresponding address
    - public void successReport – stores all addresses successfully crawled
    - public void addToVisited – stores all visited pages
- Extensions and Libraries
    - Hashtable

IV. User Instructions

To use the solution, copy and paste the JSON file to be parsed into the jsonPage string value in the webCrawl class. The solution will automatically crawl to other pages, skip duplicate pages, and mark missing addresses.

V. Testing

```
"C:\Program Files\Java\jdk-11.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2018.2.4\
Successful Crawls:
{5=http://foo.bar.com/p5, 4=http://foo.bar.com/p4, 3=http://foo.bar.com/p3, 2=http://foo.bar.com/p2, 1=http://foo.bar.com/p1}

Skipped:
{1=http://foo.bar.com/p1}

Error:
{}

All is finished!

Process finished with exit code 0
```

```
"C:\Program Files\Java\jdk-11.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2018.2.4\li
Successful Crawls:
{5=http://foo.bar.com/p6, 4=http://foo.bar.com/p5, 3=http://foo.bar.com/p4, 2=http://foo.bar.com/p2, 1=http://foo.bar.com/p1}

Skipped:
{4=http://foo.bar.com/p5, 3=http://foo.bar.com/p4, 2=http://foo.bar.com/p1, 1=http://foo.bar.com/p2}

Error:
{2=http://foo.bar.com/p3, 1=http://foo.bar.com/p7}

All is finished!

Process finished with exit code 0
```

VI. Code Repository

https://github.com/caleb-lally/GEDS-Webcrawl

VII. Reflection

The creation and development of the solution was a properly challenging endeavor that allowed me to quickly refamiliarize myself with the Java programming language. I spent a couple of days after being emailed the assignment to conduct research on web crawlers, as I had never written one before in either an academic or professional setting.

After configuring IntelliJ IDEA for proper development work, I spent time writing trivial programs in Java to better reacquaint myself with object-object oriented development, as well as Java's specific syntax. The most recent major projects that I pursued before the Ctenidae solution were either written in C or featured little object-oriented principles to begin with. After getting back to the basics, I experimented with the Hashtable data structure for a day and referenced documentation pertaining to it. Useful in its ease of insertion, deletion, and access, I found it suitable for tracking and reporting purposes, as new items could be effortlessly put into the table with a key and could be quickly displayed by simply printing the table itself to console.

The first major difficulties I encountered with the solution were entirely centered around parsing the JSON file. I had originally planned to simply read the file and use regular expressions to scrape and store the data one address and link at a time into a Hashtable, but soon discovered that my desired algorithm would be unsuitable for working with JSON. Online research pointed me to a couple of modules useful for parson JSON, but I was not able to find any documentation or tutorials that explained how to bind and work with the data to a satisfactory degree. I eventually discovered a detailed online video tutorial series that described how to deserialize a

JSON file into a Java object via the Gson module and imported Gson into Intellij via the Maven repository.

The second major difficulty encountered while developing the solution was getting the JSON to properly deserialize into a Java object so I could make use of it. I spent a period in excess of twenty-four hours searching for an explanation of the proper deserialization process before discovering an online resource that would automatically generate the correct object structure based on the structure of the JSON file. I was then able to read the addresses and links of the JSON files provided for testing purposes as strings and was at that moment able to begin work on the actual crawling algorithm.

The third obstacle overcome while developing the solution was finding the correct syntax to access the data stored in the object. I extensively experimented with setter and getter methods before noticing that IntelliJ allowed the user to view exact access syntax during debugging with breakpoints by right clicking and selecting the “Evaluate Expression” option in the right-click menu.

After being able to access and display all the required information, the fourth and final challenge was altering my algorithm for both the web crawling functionality as well as implementing multithreading. Since I had to switch technologies and structures several times during development, I was left without a viable algorithm to follow, and found it necessary to develop a new one. I experienced significant difficulty in getting each address to crawl exactly one time and developed a couple of simple methods to search for the presence of the specified address in the pages list, as well as receive its element as an integer. I was eventually able to

precisely match both test cases for both files, and then implemented a thread object for launching the methods as threads when crawling to other pages, satisfying the parallelism requirement.

SOURCES USED:

www.jsonschema2pojo.org

https://www.tutorialspoint.com/java/java_multithreading.htm

https://www.tutorialspoint.com/data_structures_algorithms/hash_data_structure.htm

https://www.youtube.com/watch?v=BbI8FdQOKNs

https://www.youtube.com/watch?v=ou2yFJ-NWr8&index=2&list=PLpUMhvC6l7AOy4UEORSutzFus98n-Es_l

VIII. Conclusion

While my solution is effective in performing according to the specified test cases, as well as implementing some multithreading functionality, it is not without flaw. Public classes and methods were extensively used over more proper private solutions, and additional multithreading could be implemented by launching each thread to run the crawlLink method as each link is passed to it. However, given the circumstances on which I received the project, I am thrilled to have delivered a working and relatively clean solution despite the difficulty, as well as providing documentation alongside the code repository. I learned a massive amount while undertaking development of the Ctenidae project, and look forward to potential future challenges to further hone my skillset.