**Web Crawler: arn8068**

**Abstract:**

A web crawler is a piece of code that travels the Internet and collects data from various web pages, also known as web scraping. Some web crawlers are autonomous and require no instructions once started. This project will focus on a user driven web crawler where user input will direct where the crawler goes and how the collected data is analyzed. Web scraping replaces the need for manual data entry and more easily reveals trends among data collected. It can also aggregate information from multiple sources into one central location. While this application provides specific examples of web crawling/scraping, it could be easily altered to better suit additional markets and/or needs. The proposed system for this project is a web scraper that can access and extract data from websites using a web application as an interface for user interaction. The extracted data is then stored in a database, as the web application allows the user to search through and query the saved findings.

**How to Run the application:**

1. Extract the contents of the .zip file into destination folder on the system.
2. Open the project in Eclipse.
3. Go to Run > Run Configurations > Arguments > Program arguments

Downloads media files i.e., images, videos etc.

Numbers of Threads

Maximum docs to download

-1 = unlimited

Graphical user interface, text, application, email

Description automatically generated

Maximum levels to crawl

-1 = unlimited

Enter the output directory

Enter the URL

1. Please enter <url> <directory> <maxLevel> <maxDoc> <maxThreads>

URL: The starting URL, including 'http://'!

directory: A filename prefix or a directory. The file name, where a file from the web is stored, is composed as this prefix + the host name + the file name.

maxLevel: The maximum number of links that are followed. E.g., if maxLevel = 1, only the starting URL and pages directly linked from the starting URL are crawler. Setting maxLevel to -1 means that the link depth is not limited! However, you can set a maximum number of documents that are saved. The default value for maxLevel is 2.

maxDoc: The maximum number of files that are saved to disk. Setting maxDoc to -1 puts no restriction on the number of files that are downloaded. If you set both maxLevel and maxDoc to -1, it is not likely that the crawler will stop by itself within reasonable time, given that you do not disconnect your machine from the web. This setting makes only sense if you decide to stop the application by pressing Control+C later. The default value for maxDoc is -1 (unlimited).

maxThreads: The maximum number of threads that are used to crawl the web. The faster your own connection the more threads you can sensibly use. Depending on the speed of the remote host, using DSL connection with 768 KBit/s downstream, up to ten threads are appropriate. For that reason, the default value of maxThreads is 10.

**Architecture:**

Shape

Description automatically generated with medium confidence

**Technical Description:**

Web crawling speed is governed not only by the speed of one’s own internet connection, but also by the speed of the sites that are to be crawled. Especially if one is crawling sites from multiple servers, the total crawling time can be significantly reduced, if many downloads are done in parallel.

1. **Processing items in a queue:**

Web crawling can be regarded as processing items in a queue. When the crawler visits a web page, it extracts links to other web pages. So, the crawler puts these URLs at the end of a queue and continues crawling to a URL that it removes from the front of the queue.

It is obvious, that every algorithm that just works by processing items that are independent of each other can easily be parallelized. Therefore, it is desirable to write a few classes that handle multithreading that can be reused.

Java provides easy-to-use classes for both multithreading and handling of lists. For multithreaded web crawling, we just need to enhance the functionality of Javas classes a little. In the web crawling setting, it is desirable that one and the same webpage is not crawled multiple times. We therefore do not only use a queue, but also a set that contains all URLs that have so far been gathered. Only if a new URL is not in this set, it is added to the queue.

1. **Implementation of a thread controller:**

As mentioned above, Java has a good interface for handling threads. However, in our special case, we can add a little more generic functionality. In our case, we want to make use of several threads that process items from the queue. A 'controller' should create new threads, if and only if there are still items in the queue to process and if the total number of threads does not exceed an upper bound.

In our implementation of such a thread controller, we provide the controller class on construction among other parameters with the class object for the thread class the queue. The queue should be pre-filled with at least one item. We require from our thread class, that it implements the run method it inherits from the Thread class.

1. **Implementation of a thread:**

We expect from the thread in the run method, that it fetches new items from the queue, and that it ends itself if there are no items left. This is common for all our possible threads; therefore, we can implement this in an upper class for 'controllable' threads.

If there are no more items to process, the Thread can terminate itself, but must inform the Thread Controller about this. In the web crawler scenario, this is important when all URLs for the link depth are processed, and the next deeper level is reached. The Thread Controller is now responsible for shifting the queues and starting new threads as needed.

1. **Messages:**

During program execution, it might sometimes be necessary that a 'working thread' tells the 'main thread' what it does now. It the web crawler application e.g., the user might be interested in what page the crawler is currently visiting. The main class can register itself or another class as a message receiver. The message receiver class is then notified, when a thread does something, a thread is terminated, or all threads are terminated. In the object-oriented paradigm calling a method is equivalent to sending a message.

Flow chart showing the webcrawling
algorithm