Overview

The project aims to develop a multithreaded web crawler in Java, which will explore a website starting from an initial URL, gather all reachable HTTP links, and recursively download content from each link up to a specified depth. The crawler will run each link download in a separate thread for parallel processing, enabling efficient and fast crawling. Additionally, the crawler will collect and log essential information about the links it visits, such as the URL, the size of the content, and the path of the link, and present this data in a browsable index. And try to shows these with interactive GUI.

Objectives

1. **Design a Multithreaded Web Crawler:**
    - Implement a recursive process that crawls web pages, extracting links and following them to gather additional pages up to a given depth.
2. **Parallelize the Downloading Process:**
    - Use Java's multithreading capabilities (e.g., `Thread`, `ExecutorService`) to download the links in parallel, reducing the time required to crawl multiple pages.
3. **Information about Crawled Links:**
    - Maintain a table that tracks URLs, their sizes, and paths to each of the links.
4. **Develop a Browsable Index:**
    - Create a browsable interface GUI-based that lists all crawled links and displays associated data.

Technical Approach

1. Web Crawling

- **Starting URL:** The crawler will start from a given URL and parse the content to extract all HTTP links. For this, we will use `URLConnection` and `HttpURLConnection` to connect to and download the webpage content.
- **Link Extraction:** Once the content is downloaded, we will parse the HTML to extract all the links. This can be achieved using regular expressions or HTML parsers like `Jsoup` (third-party library) to extract links from the `<a>` tags.
- **Depth Limitation:** The crawler will follow the links recursively up to a specified depth to avoid going too deep into the web. For example, if the depth is 2, the crawler will follow links on the starting page (depth 1) and then follow links from those pages (depth 2).

2. Multithreading

- **Thread Per Link:** Each new URL found during the crawling process will be assigned to a separate thread for downloading. This will enable parallel downloads of multiple pages.

- **Thread Pooling:** Instead of creating a new thread for each URL, we will use `ExecutorService` with a thread pool to manage concurrent tasks efficiently.
- **Synchronization:** Proper synchronization will be implemented to handle shared resources like the log file to ensure thread safety.

3. Information Collection

- **Log Structure:** We will collect key data such as:
  - URL: The address of the page.
  - Size: The size of the page in bytes.
  - Path: The full path of the URL, representing where it exists in the directory structure.
  - Crawl Time: The timestamp when the page was crawled.
- **Log Storage:** Use mysql database to store data.

4. Browsable Index

- **Interactive GUI:** A hierarchical GUI can be created that will allow the user to view connected websites.

Tools and Technologies

- **Java Core Libraries:**
  - `URLConnection` and `HttpURLConnection` for making HTTP requests.
  - `ExecutorService` for managing the thread pool.
  - `Thread` and `Runnable` for basic thread management.
  - `File` and `IOException` for file operations.
- **HTML Parsing:**
  - `Jsoup` (optional) for extracting links from HTML pages.
- **Data Storage:**
  - `Mysql` **database.**
- **Interactive GUI:**
  - `Swing` for building a web interactive GUI.

Steps

1. **Step 1:**
   - Research and familiarize with the Java `URL` and `HttpURLConnection` classes.
   - Set up the basic project structure and implement a basic version of the web crawler that can crawl a single page and extract links.
2. **Step 2:**
   - Implement multithreading using `ExecutorService` to handle link downloading concurrently.
   - Start implementing the database functionality to store URL data.
3. **Step 3:**

    - Add recursion to follow links up to the specified depth.
    - Test the crawler with different depth settings and ensure it handles multiple threads efficiently.
4. **Step 4:**
    - Implement the browsable index from database to interactive GUI.
5. **Step 5:**
    - Conduct testing, handle edge cases (e.g., invalid links, timeouts), and optimize performance.
    - Finalize documentation and project report.

Expected Challenges

1. **Handling Large Websites:** Crawling large websites with many links could lead to performance issues. Thread management and efficient data handling will be key.
2. **Error Handling:** Ensuring that the crawler handles HTTP errors (404, 500, etc.), timeouts, and other network-related issues gracefully will be crucial.
3. **Concurrency Issues:** Proper synchronization will be required to avoid issues like race conditions when accessing shared resources (e.g., the log file).

Conclusion

This multithreaded web crawler will efficiently crawl websites, download pages concurrently, and provide useful metadata about the pages. Utilizing Java's networking and multithreading features, this will ensure that the crawler is fast and scalable. The project will also help me understand various concepts such as concurrency, HTTP protocols, and database, which are essential in building high-performance networked applications.