1 Overview

1.1 Function

This extension can be used for a) searching over linked pages of a given page and b) performing topic analysis on these linked pages.

For each linked page, its text would be scraped and stored as a document, then an inverted index would be built from the collection of documents. Users can perform query search or topic analysis using the inverted index.

1.2 Components

The program consists of three componenets.

1.2.1 Chrome Extension Frontend (JavaScript)

It provides basic UI elements for interaction and input. Users can enter query or adjust parameters for topica analysis. It collects the inputs and the URL of current page, then send to the Python backend server through WebSocket. There are three main operations: "Build", "Search" and "Topic Analysis".

1.2.2 Python Scraper/Backend Server

Build: It scrapes the given URL and store all linked pages on disk (text only, non-recursive). Then invokes the Java backend program to build index.

Search/Topic Analysis: It invokes Java backend program to search/analyze topic.

It also maintains metadata of documents, e.g. URL and link text

1.2.3 Java Backend Search Engine

It supports three operations:

Build: Given a directory of documents, create an inverted index using Lucene (a Java library providing powerful indexing and search features).

Search: Assume an inverted index exists, perform query search and output the top results.

Topic Analysis: Assume an inverted index exists, perform topic analysis (PLSA algorithm) and output the topic for each document.

The Java backend program can work independently without other two components.

2 Implementation Details

2.1 Chrome Extension Frontend (JavaScript)

popup.js/popup.html: On "Start" button clicked, retrieve the URL of current tab and send it to background.js.

background.js: When a URL from popup.js is received, open a new tab (search.html) and maintain an tab id to URL dictionary.

When a message (an object that specifies operation to be performed and related arguments, e.g. {"query": "test"} means the user want to search for "test") from search.js is received, look up for the corresponding URL, attach it to the object (i.e. {"query": "test", "url": "https://xxx.org"}), convert the object to string (JSON.stringify()) and send it to the server through WebSocket.

When a response from the server is receieved, pass it back to the search.html tab.

search.js/search.html: Send the user inputs as an object to background.js, and display the response from the server.

2.2 Python Scraper/Backend Server

server.py: A WebSocket server that listens for messages from the frontend, invokes the scraper or the search engine to handle requests.

scraper.py: Given a URL, scrape all the linked pages and store them on disk by calling file_manager.FileManager.save().

file_manager.py: Maintain the metadata of documents. ".meta" records (URL, directory name) entries. For each directory, ".toc" records the link text, URL and file name.

search_engine.py: A wrapper of the Java search engine. It creates a `subprocess` of the Java search engine. Since the Java search engine does not have information about where the documents come from, search_engine.py would look up the metadata from FileManager add back the URL or original link text to the search results.

2.3 Java Search Engine

2.3.1 Source Code

The search engine is mainly impleneted using Lucene.

Main.java: The main function.

Indexer.java: Build the inverted index using Lucene.

Searcher.java: Search the inverted index using Lucene. It also makes use of Lucene Highlighter to highlight the matching keyword in search results.

PLSA.java: PLSA algorithm with background distribution. The word counts information is retrieved from the inverted index. The EM algorithm is implemented in simple for loops with no optimization.

2.3.2 Compiled Executable

main.jar: The compiled executable jar file, can be called by **java -jar main.jar [args...]**

Expected output:

```
For building index: -b [directory path]
For searching: -s [directory path] [query]
For topic analysis: -t [directory path] [number of topics] [probability of
background]
```

It assumes the collection of documents is stored in **[directory path]/contents**, the inverted index would be stored in **[directory path]/index** after build. A background word counts file **.background** is needed.

3 Instructions

3.1 Requirements

1) Python 3

2) pip install tornado bs4 requests

3) Java SE 17 (https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html)

3.2 Steps

1) In Chrome Extension page, toggle developer mode and load unpacked, select **extension** folder.

2) In **server** folder, run command **python server.py** to start the server.

3) On the target page, click the extension then click the "Start" button, a new tab should be created.

4) Build index (it may take few minutes).

5) Enter query to search or analyze topics (topic analysis may take few minutes, depends on number of docs and number of topics).