Assignment 3: Web Searching

The purpose of this assignment was to build a mini search engine which searches and ranks pages based on the PageRank algorithm.

Program Structure:

The solution consists of four java files:

- **LinkExtractor.java**: This class file consists of functions to parse through the file of links (i.e. test3.txt). The functions expect links to be present in the form **<number, link>**. There are functions to build the matrix to denote the connections between the different pages and this information is then used by functions in RankPage class to compute PageRanks. Also, while parsing the links, the class prepares index information for each page. The class functions use the **jsoup** library to parse HTML pages and extract relevant data.
- **RankPage.java**: This class file consists of functions to compute PageRanks for all the links that are parsed by functions in the LinkExtractor. The functions in this class build upon the matrix constructed by the LinkExtractor class and use the PageRank iteration approach to reach upon the PageRank for each link. The class also consists of functions to read and write the metadata file holding the index record for each link. The class uses the JAMA (Java Matrix) package to perform matrix computations.
- **PageInfo.java**: This class serves to give structure to the index information about each link. It consists of information about the page title, link, PageRank and anchors to the page and functions to set and get these values.
- **SearchEngine.java**: This class consists of functions to handle the user's query. It constructs an index upon the metadata gathered while parsing the links and computing the PageRank information. This class allows the user the ability to either parse all the links and rebuild the metadata and then take user queries or use existing metadata information on the disk to build the index and accept user queries. Such an approach makes the system more flexible because if the links do not change then the time taken to parse them again and again is saved.
  In order to run the entire system, run this class and provide the required attributes as described in the following sections.

Solution Explanation:

There are two constructors in **SearchEngine**. If the user intends to rebuild the index using an existing metadata file the constructor taking only one parameter (i.e. the metadata file path) can be used. If the user wishes to rebuild the index then the constructor accepting two parameters (i.e. the path to the links file and path to the directory where the metadata file (metadata.txt) is to be created) is used. Decision about which constructor is to be invoked depends upon the number of parameters passed by the user at runtime. Execution instructions will be covered in the next section of this report.

When the constructor of SearchEngine accepting two parameters is invoked, the constructor for **RankPage** is invoked with the path of the links file. RankPage in turn calls the constructor for **LinkExtractor** with the file path and invokes the **parseLinks** function. parseLinks goes over all the links in the links file and constructs a matrix representing connections between the different links. This matrix is then normalized in RankPage by calling the **normalizePageConnectionsMatrix** function. After this, the **generatePageRanks** function is invoked from SearchEngine and it is here that the PageRanks for all pages are computed iteratively using the formula:

$$w_k = dBw_{k-1} + (1 - d)z_0$$

Here w_k represents the page weight matrix in the next iteration, w_{k-1} represents the current page weight matrix, B denotes the normalized link matrix, z_0 denotes a vector with all values set equal to 1 and d denotes the damping factor. For the purpose of this assignment the value of d is set at **0.85**.

The **hasConverged** method checks to see if the iteration has converged or not. Regular computations continue until the result converges. Now, that ranks are ready, the **generateMetadata** function is invoked and it is here

that all the required information is pushed into a file called **metadata.txt** at the directory path provided by the user. The metadata.txt file consists of the data in the following format for each link:

```
<Link number>:<Link Title>
PageRank:<PageRank value>
Link:<Link address>
Anchors are as under:
<List of anchors in the form <Link number>:<Anchor text>>
**********************************************************************
```

Please note that here Link number denotes the number assigned to the link while sequentially reading the file and not the numbers as mentioned in the file.

While parsing the links in the LinkExtractor class, a mapping is kept between the link number and the relevant page information by using a TreeMap called **pageInfoMap**. The key there is the link number and the value is an object of the class PageInfo which covers all information relevant to the link in the key. This pageInfoMap is the one which is then iterated over and pushed into the metadata file.

Once the metadata information is ready, SearchEngine now constructs an index over this information by invoking the **prepareIndex** method. The prepareIndex method iterates over all the anchor texts and titles for each page as collected in the metadata. It uses space as the delimiter in each case and breaks up the title and anchor string text into tokens. A TreeMap called **index** is then prepared which is a mapping between a word and a list of all link numbers that the word appears in. Now, when a user's query comes in the form of a single word, the corresponding list of link numbers is extracted from the index by supplying the word as the key to the get function of the map. The PageRank corresponding to each of these links is then collected by invoking the **getPageRank** method of PageInfo and links are then listed in descending order of the PageRank along with their title, PageRank value and some content from the page. **displayContent** method is invoked to generate a snippet from each link. This snippet is limited to about 20 words (defined by **snippetSize**) around the user's search term. If the search term does not appear in the body of the page, then the first 20 words are displayed.

If the constructor for SearchEngine with one parameter (i.e. path to the metadata file) is invoked, then the **readMetadaFile** method of RankPage is invoked and the pageInfoMap as described earlier is constructed directly from the metadata file. The query handling part remains the same i.e. **handleUserQuery** method is invoked which uses the constructed index to respond to the user's query.

Assumptions:

- If the query consists of more than 1 term, then the subsequent terms are ignored and results are provided only for the very first term in the query.
- Query 'ZZZ' is used to exit the system.
- The index prepared does not account for multiple occurrences of the same term referring to the same link. If a single term is used to refer to the same link multiple times, then it is only accounted for once.
- The file consisting of all the links to be crawled always consists of links in the form **<number, link>**.
- The system requires a persistent internet connection to function correctly.
- If a line in the links file does not consist of a link then it is ignored altogether.
- If a link does not consists of links to other pages, then all the values in the column representing that link in the link matrix is set to 1/n where n is the number of links in the system except the value representing the connection to the link itself. That value is set to 0.
- No results are returned if the user enters a term which does not appear in the index.
- All links relevant to the query are displayed.

Execution Instructions:

Unzip **a3.zip** to get Report.pdf (report file), SampleRuns.pdf (sample runs file), metadata.txt (metadata file), Binary and Source folders. The Source folder consists of the src (consisting of the minisearch package which

contains LinkExtractor.java, RankPage.java, PageInfo.java and SearchEngine.java) and bin (consisting of the minisearch package which contains LinkExtractor.class, RankPage.class, PageInfo.class and SearchEngine.class) folders.

Inside the Binary folder there is a file called MiniSearch.jar. If the user wishes to reconstruct the index using the links file then go to the command prompt and type the following command:

> java -classpath **<path to MiniSearch.jar>** minisearch.SearchEngine **<file path on system where the links file i.e. test3.txt is kept> <directory path on system where the metadata file i.e. metadata.txt is to be created>**

If the user wishes to use an existing metadata file then the following command should be used:

> java -classpath **<path to MiniSearch.jar>** minisearch.SearchEngine **<file path on system where the metadata file is kept>**

Thus, depending upon the number of parameters passed here, the system is able to decide which constructor of SearchEngine is to be invoked as described in the last section.

Note: Please use jdk 6. It is available on machines in the CSUGLAB

Sample Output

Details of the different test runs are covered in SampleRuns.pdf . Please refer to that document.