COM4280 : Intelligent Web

Assignment 1, Group BDM
Kushal D'Souza (aca08kd@sheffield.ac.uk)
Shaabi Mohammed (aca08sm@sheffield.ac.uk)
Zsolt Bitvai (eca08zb@sheffield.ac.uk)

Introduction

The web spider is a piece of software that methodically browses the world wide web. The BDM spider we have implemented has two main components which are the web crawler and the document indexer. These can be run using either the graphical user interface or a command line interface.These components have been implemented in such a manner so as to allow the user to use the indexer and crawler in a single application or as separate applications. How this has been achieved will be illustrated later in the report.

The web crawler takes in as its input a valid web URL (Uniform Resource Locator). This is the base URL. It then scans the web page that this URL points to in order to identify all the URL's are located on this page. What this means is that all the URL's that this current page links to will be stored in the memory of the crawler. It then analyses these URL's found in order to find URL's that they link to. This crawler obeys the robot.txt file in such a manner so as to not process the files that are disallowed by the robots.txt. All valid and allowed URL's that are found internal to the base URL are written to separate files, which are read in subsequently for indexing. All other types of URL's are discarded and are not indexed. The working of the indexer is explained below.

The indexer takes as its input a file containing a list of URL's, one URL per line. It then reads the text contained on the web pages that each of these URL's point to. The words contained in the text are then used to create an index which can be used to search the list of URL's. The index stores a word and a list of URL's that contain this particular word. The indexer allows a user to write the index to a file and load an index from a file as stated by the system requirements. This report will elaborate further on the workings of our web crawler/indexer, the design choices we made and the issues we faced during the implementation stages.

Task

User Interface

The API design of the Crawler and Indexer Allows the GUI to be extensively designed in a modular and flexible manner. The GUI for the application has three functioning modes, allowing the user to select and customize the way in which the application would function.

The first functioning mode for the GUI is the Full Application mode, the full application mode presents the user with a user menu where they can select from a menu either of the Crawling, Indexing or Search engine features.
The second GUI mode is invoked by providing the applications with appropriate command line options (explained in Design choices). This application allows the crawler, indexing and search engine to run as a standalone application.
Lastly, the software also runs in a GUI less command line mode allowing the user to run the application simply as a client application, and takes in all required input as command line arguments.

Web Crawler

The crawler access all types of pages that are parsable, meaning it has a content MIME type of “text/*”. These include among other text files, html, XML and txt. It is implemented as a generic crawler so it doesn’t have any site specific configuration except for reading the robots.txt file from the assignment webpage, as it is not situated in the root directory. It has meaningful headers to identify itself including a user-agent header, set to ”BDM_crawler_University_of_Sheffield_COM4280/1.0” as well as it accepts languages in English and has a proper application content MIME type. The spider can be controlled both from the user interface as well as the command line. It can be started with an initial seed, paused, resumed and stopped. If the spider finishes normally or if it is stopped it prints all the URLs it has found in separate files in “Output/Spider/*url-type_of_link*.bdmc”. These links include all local, external, dead, non-parsable and disallowed links it has found. In order to determine whether to parse a URL, the spider checks the robots.txt file. It parses the relevant entries and respects all rules. When accessing robots.txt it checks the crawl-delay value and respects that. It only maps the given site structure and does not follow external links. It keeps track of all the links found and does not visit the same link more than once. In addition the Crawler implements an API interface for other classes to access.

Document Indexer

The document indexer takes as its input a file containing a list of URL’s. It then retrieves the web pages associated with each of the URL’s. The indexer downloads all the data contained on the page and then a parser is used to de-HTML the page. What this means is that all the HTML tags from the data are removed along with the special characters, CSS & JavaScript. Thus the parser only returns the text contained in the document. This text is saved as a String of words separated by the white space character. The words are then stored in the memory along with the respective URL’s from which they were obtained. This list of words along with the URL’s is used to create the inverted index. A function has been provided which allows a user to write the index from memory into a file.

This follows the following format:
<keyword> URL URL....
<keyword> URL URL...

The keyword is written to a line followed by a space. Then all the URL's that contain this keyword are written to the same line each separated by spaces. Once all the URL's for a particular keyword are written to the file, the index for the next keyword is written to the next line. This index can later be loaded from the text file back into the memory. The implementation of the indexer also allows the user to search through the list of URL's for URL's that contain a particular keyword. On running a search for a particular keyword, the implementation asks the user to load the index from a text file into memory. It then searches through the index and returns a list of URL's that contain the keyword being searched for.

Issues

We have used GIT for source control. It proved to be difficult to set up and a significant amount of time was spent as there were issues in merging and commit to git. However, despite these issues it worked to our benefit as it allowed us to collaborate on the project implementation.

User Interface

It was a little trivial on deciding the best way of how the API would function and interact with the graphical user interface and the command line interface using a singular unified method. This was later solved by adapting the ActionListers to react differently and depending on which software mode was selected, therefore actually simplifying and speeding up the programming process in a standard manner.

Web Crawler

The main issues with the web crawler were sorting out the stop, resume, start and pause functions as they are invoked with different threads so as not to freeze the user interface. In addition, as no 3rd party libraries could be used, complete robots.txt and html parsers had to be written to process the robots.txt file and webpages. In the HTML parsing we had some issues identifying all links on a page, which included checking various attributes of tag. Last, but not least, the printing had to be correctly synchronized with all the various states of the crawler. Deciding when to print and when no to print, as well as making sure we have permission to write files and the directories exist.

Document Indexer

The main issues faced in the implementing of the indexer were the parsing of the text from the HTML of the page. We used the Swing HTML Editor toolkit to parse the text. However, this did not completely remove the special characters and unnecessary white spaces from the text. We had to write regular expressions in order to filter out just the text. One of the other difficulties that we had to tackle was in the implementation of threads. Since the GUI, the web crawler and the indexer all run in separate threads, a considerable amount of time was taken in order to implement the pausing, starting and killing of the indexer.

Design choices

The multi threaded model has an advantage over the single threaded model, because it enables the user interface to function when background operations take place. In addition, by strictly separating the functions of the program: user interface, web crawler, document indexer, we are able to selectively load modules. That means the user interface has got a global interface, one for the web crawler and one for the document indexer. The web crawler and the document indexer can also function separately via the API or the user interface. To notify the user interface the web crawler and document indexer uses the Observer patterns, and also implement thread synchronization to allow the results to be updated seamlessly. All modules of the software have been put into separate packages.

The requirements have been flexible on the implementation details as well as the design choices. The only requirement was to have the API implemented for the crawler and the document indexer. This is done by implementing the required interfaces.

User Interface

The software can be run using the Graphical User Interface as well as through command line.The software can be run through various command line options which are specified below, these command line options

The two commands shown below run the application in full application mode.

RunSpider

RunSpider -g

This command runs the application in standalone Crawler GUI mode

Runspider -g c

This command runs the application in standalone Indexer GUI mode

Runspider -g i

This command runs the application in standalone Searchengine GUI mode

Runspider -g s

This command runs the application in Command line crawler mode

RunSpider -cli c <websiteURL>

eg : java Runspider -cli c http://poplar.dcs.shef.ac.uk/~u0082/intelweb2/

This command runs the application in Command line indexer mode

RunSpider -cli i <input URL list file>

eg : java Runspider -cli i ./output/spider/poplar.dcs.shef.ac.uk_externalIWURLs.bdmc

This command runs the application in Command line indexer mode

RunSpider -cli i <indexed keyword database> <search keyword>

eg : java Runspider -cli s ./output/spider/poplar.dcs.shef.ac.uk_index.bdmi toll

Web Crawler

The crawler has an interface called Crawler which implements the myIWSpider interface. This communicates with the user interface too through the Observer pattern. This means whenever something changes in the Crawler state, the user interface is notified and updated stats are fetched. The spider functions are implemented in SpiderImpl class. This uses two classes, called Parser for parsing html pages and Links for keeping track of various types of links. The class is initialized with a base site url this it fetches and maps the site structure of. Then it reads the robots.txt file and sets up the rules for disallowed links. When the Crawler is started, the base url is added to the queue. The queue is then processed one url at a time. Each url is checked that it is external, disallowed, non-parsable, local or dead. If it is local, it is parsed and the links from the parsed url are added to the active link queue for processing. If the spider is stopped or it finishes normally, it prints all the links to separate files. The spider can be stopped and resumed, started or killed. It functions with a user interface and without.

One limitation of the spider is that it needs to open a connection to check the header of the url to read the content type. This may take a while, if the connection is timed out, the link is identified as a dead link. The spider filters out css and javascript, however if a website is a coding blog and discusses javascript and css, then relevant files could be ignored by the parser. Also, if a link is included in the text as plain characters, then it is not recorded by the spider and thus is not subsequently parsed.

Document Indexer

The indexer is designed in the manner described below keeping in mind the requirements specified. The indexer implements the myIWSearchEngine interface as stated by the requirements. Thus, it overrides the openInterface,closeInterface, IndexCrawledPages,search and the loadIndexTable functions. The IndexCrawledPages function takes in an input file name and an output file name as its parameters. On calling the IndexCrawledPages function, the indexer is started in a new thread after which it begins to parse the contents of the site pages and make a list of keywords from the contents. An index is created using the keywords which is then written to the corresponding output file. The indexer is only allowed to take in files that have a .bdmc extension as the input and the output is written to a file with a .bdmi extension. The filename is chosen automatically. The indexer also takes into account the number of requests it can make per second as specified by the robots.txt file.

The other functions that the indexer implements are the search and the loading of the index from a text file into memory. Initially we had planned to implement these functions in such a way that they would be run in separate threads. However, this was made difficult because of the way in which our classes interacted with each other. According to the requirements, the search function is supposed to return the results obtained by the search. However, if we implemented this using threading, we would only be able to print out the search results to the GUI and not return them. If a user wants to search for a listing of URLs for a particular keyword, he will be asked to load an index from the text file into the memory. The keyword is saved in a HashMap so as to allow us to easily obtain the set of URLs that map to the particular keyword. The HashMap maps keywords to the set of URLs. This is then displayed through the GUI. We have used the Set object to store search results. The reason we used a Set is because it removes duplicate entries as opposed to an array.

One of the limitations of the system we have implemented is that the search feature only takes in one keyword as its input. We have also not implemented a ranking algorithm. The HTML parser is coded so that it first removes all the HTML tags and then checks for any special characters and remaining HTML tags present in the text. If these are present, they are removed from the page. Thus if you have a web page that for example, contains programming tutorials on the pages, the keywords obtained from these pages will be stripped off some of the special characters used in the syntax of the programming language. Thus, the search in turn might not return appropriate results. URLs present in the text and that are not hyperlinked will also be removed and these will not be indexed.

Conclusion

The crawler works effectively on all web pages taking into account the robots.txt file. It doesn't create DOS attacks because it waits between requests. The user interface is extremely flexible and gives useful statistics to inform the user.Overall, the implementation of the system is very robust. It can be operated using both the Graphical User Interface and the command line. The fact that the spider has been implemented using threads allows the crawler and the indexer to run simultaneously provided that the indexer has an input file to read from.

Division of work

We have divided the work equally between the three members of BDM team. Initially, during the planning stages, we discussed about how we would implement the Crawler and Indexer and set areas that each of us would overlook. Shaabi Mohammed was responsible for the user interface and linking to the api, Kushal D'Souza for the document indexer and searcher, and Zsolt Bitvai for the web crawler. We needed to set responsibilities to make sure everything is finished before the deadline, however we collaborated with each other during the implementation stages by helping each other when we got stuck with a hard problem and also did extensive pair programming at times. This included figuring out the implementation and design choices together through face to face meetings and instant messaging and voice conferences. Also we reviewed each other's code to make sure the software is reliable and robust. Every member of the team was familiar with every aspect of the assignment and through efficient coordinated team work we have managed to finish the implementation as well as the documentation on time.

Extra Information

The software is packaged in a zip file, extracted folder will be recognised as a project in both eclipse and netbeans.
To compile and run the software the following sequence of commands can be used

```
cd aca08sm
javac webspider/RunSpider.java
java webspider/RunSpider
```

Bibliography

During the implementation of the spider, we have used ideas and source from Jeff Heaton's

crawler for the basic spider implementation and then built on this.
http://www.developer.com/java/other/article.php/1573761
http://www.jeffheaton.com/java/bot/javaspider.shtml

In order to remove the HTML tags from pages, we had to use regular expressions. We have used information from stackoverlow and the sites listed below to help in creating the regular expressions.
http://stackoverflow.com/questions/1732348/regex-match-open-tags-except-xhtml-self-contained-tags
http://stackoverflow.com/questions/181095/regular-expression-to-extract-text-from-html
http://www.regexlib.com/REDetails.aspx?regexp_id=153

In addition, team has used for making the right design choices for the spider. www.google.com