**Abstract**

The World Wide Web or WWW is a huge collection of scattered data. All information these days can be uploaded to the internet and can be retrieved easily with the help of a search engine. Each website is basically a huge chunk of data stored in a document and uploaded to a server. In order to retrieve and view data on several websites, we use search engines. Search engines display the searched data by using several Search Engine Optimization (SEO) algorithms. One way to successfully traverse data is by the usage of web-crawlers. Web-crawlers efficiently traverse each web-document, stores their data using web-scraping, indexes each page and efficiently show data to the user in less time. In this paper, we have proposed a multi-threaded web-crawlers based on a unique combination of data structures and programming techniques such as priority queues, arrays, sorting algorithms etc. Our paper shows a way that uses priority queues based on nested linked lists that sorts our data word-wise and then the queues within the queues sorts the URLs linked with a particular node/word.

**Introduction**

Web spiders or spider generally known as web-crawlers and web bots are programs written by companies such as Google, Yahoo, Bing to traverse several web-pages and for the retrieval of information on different web-pages. Web-crawlers searches each website and uses several hyperlinks for indexing the data. Web-crawlers are always used with web-scraping. Web-scraping is a method of storing data available on the World Wide Web (WWW). Web-scraping is used by a web-crawler to extract huge chunks of data from every website and store that data on a local database. The extracted data is stored along with an index which is then further used in order to sort the data according to the retrieval rate of the users around the globe. The data available on the internet is either unstructured or is available in a variety of formats. There are no particular data models available for the websites to follow or there is no generic way of portraying your data online. So, one of the major tasks of a web-crawler is to add a structure to our data and to make searching, traversal faster and resource efficient.

With each passing day, the World Wide Web is increasing and more content is being added to it. Thus, searching the entire web in future would be difficult as compared to now. So, it is essential to devise a way that would make sense out of the widespread data. Research is still being done in this domain and companies with the help of data scientists are trying to develop algorithms that would not exhaust our hardware and could also maintain previously saved data instead of storing it again and again.

The key functionalities of all web-crawlers are same but they differ in the algorithms they use for sorting and retrieving the data. The main aim of a search engine is to provide pages and links that are relevant to the query of the user. Major architectures followed by web-crawlers are breath-first search and best-first search. Both are used by GoogleBots etc. and both have downsides. But, currently the most popular bot i.e. GoogleBot is used by 74.54% of the internet users.

**Literature review**

We read around 4 research papers and the review is written based on our understanding from the papers. Several web-crawlers available and used are explained below:

1. N. Singhal et. al.[1] designed a web-crawler based on incrementation. It has a database that updates its database after visiting the entire internet periodically. The web pages are categorized as very frequently and frequently less frequently. The websites are visited frequently and the number of visits to the site is changed according to the category of the website. Mainly used in parallel web-crawling.
2. S. Ganesh et. al.[3] put forward the idea of a web-crawler based on ontology. It uses an association metric. The association metric is dependent on the domain of the ontology. Based on the domain, association metric analyses the semantics and the contents of the URL. Metrics depends on relevancy too. The relevancy of every page is calculated and reordering of the metric is also done via the relevancy metric.
3. D. Mukhopadhyayet al.[4] made a web-crawler that only crawls the web-pages specific to its domain. It also uses ontology of a domain.
4. X. Chen et. al.[5] proposed the idea of a focused web-crawler. Focused web-crawlers use the content of a web-page and it focus on a structure that links the content to maintain a relevancy in traversal. Basically, all data and web-pages containing a specific data are kept together.
5. D. Hatiet. al.[6] proposed the idea of an adaptive web-crawler. What an adaptive web-crawler does is that it crawls based on the analysis of a link. Each URL has a score linked with it and a relevancy score too. The relevancy score along with the description and similarity score helps in calculating the score of an unvisited URL. It is then saved and retrieved based on that score.
6. Thenmalaret. al.[7]’ s crawler is a combination of focused crawling but the focused crawling is based on ontology. They use topic wise vectors as well as concept wise vectors. The web pages are seeded depending on the concept wise vectors. The conceptual rank of each URL is also specified in each URL.

**Methods**

Our web-crawler is a unique priority-queue based algorithm. For the purpose of this project, the scope of our web-crawler is only one text file that contains URLs, total number of times a URL has been accessed and the content in that URL.

The flow of our program is as follows:

Reads data from the file

Tokenizes the data

Stores the tokenized data along with the sum of everyword

Asks user to search

**Does the searched data match the content of any URL?**

Display Not Found message

**No**

Increases word count and access count of the URL

Changes priority in the queue of linked lists

Updates data in the file

**Yes**

Search again?

**No**

Explanation

1. First the file containing the entire data is read from top to bottom. Here multi-threading is implemented. Multi-threading helps to simultaneously read different parts of the file.
2. The data from that file is then stored in a structured array. The array of structures stores each entry of the data according to the URL. Each URL has a specific entry along with its content and the number of times it’s been accessed.
3. The content of each URL is then tokenized and stored in another structured array. This structured array stores the total number of times someone has searched that data and that word.
4. The user is then asked to search for a specific term. If the term is in any of the URLs then the access rate is increased and the number of times that term has been searched is also increased. Multi-threading is also implemented here. Different threads are assigned to different parts of the array. For example, top 10 indexes are searched by the first thread and similarly next 10 indexes are crawled by the second thread and so on.
5. After this, a linked-list based priority queue is implemented that has linked-list based priority queues on each node. This nesting helps in increasing our search speed. The main linked-list based priority queue is based on the words and their priority. Each word has a linked-list based priority queue that sorts the URL based on the number of times it has been accessed.
6. After storing the data in a linked list, next whenever we run or search another term, instead of traversing the array, we simply use linked lists.
7. In order to retrieve all of the web-pages from a specific term, we simply go to the linked lists of that word.
8. In order to get the most searched word, we retrieve the head of the linked list.

Pseudo Code:

Step1: StoreFromFile(FileName)

Step2: populateArray(AccessRate, Content, URL)

Step3: populateWords(name, count)

Step4: SearchTerm(term){ MultithreadingSearch(term, threadID)}

Step5: if(term==Array[i]); AccessRate++ && count++

Step6: populateLinkedList(words[i].name, words[i].count)

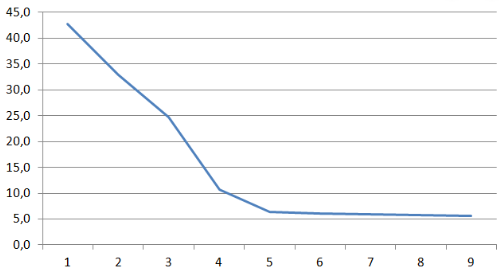
Step7: ask the user to search again

Step8: search and display statistics based on the data from linked lists

Step9: display results on GUI

**Experiments**

Our initial code was based on simple arrays and hard-coded data. Since, our code is time elapsed so it gave fairly less time in crawling the data. For example, the time elapsed for an array of 10 indexes was 0.0000000001 seconds. But as we increased our size of arrays our time elapsed increased to 1 minute. Similarly, on increasing the data size our time elapsed kept on increasing. To overcome this problem, we implemented a combination of linked lists and arrays to sort the data. Then our time elapsed became significantly smaller. Here is a representation of our data after implementing it on several files and different types of content:



Data crawled

Time Elapsed

**Tools and Programming Language**

The following tools and programming language were used for the development of the web-crawler.

1. Oracle VM VirtualBox

Since a Linux operating system had to be run on a Windows machine, the right software was required to allow one operating system to host a guest operating system on it. Therefore, Oracle VM VirtualBox was used which emulates an operating system environment on top of another one, which was Linux on top of Windows in this case.

1. Ubuntu 14.04

As the entire course of Operating Systems focused on Linux environments, the authors were focused on a Linux-based operating system for the development of the programme. Furthermore, since all the labs were also conducted on Ubuntu 14.04, it was decided to use this operating system.

1. C++ Programming Language

Initially, C programming language was used. However, due to its limited functionality and very basic features, there was a shift to C++ programming language. C++ is very similar to C in terms of syntax and coding. Nonetheless, the powers and features offered by C++ far exceed that of C which was the main motivation behind shifting from C to C++. That being said, it is important to mention that not much of the code had to be revamped since both languages are very similar. Only the additional features such as strings and nodes for linked lists were used.

**Work Division**

The entire project was a collective effort of both the authors. Right from the logic building to coding the basic skeleton of the programme, from updating the programme to realizing the need for a shift in the programming language, and from fixing the errors to the implementation of the Graphical User Interface (GUI), both the authors did every part of the project together.

However, at times, the errors were resolved separately, which include the resolution of the string-based errors by Tatheer Zahra and the node-based errors by Nayyara Naseer.

Apart from this, the logic, coding, development, documentation, and the presentation are all done while working together.

**Major Problems Encountered**

Certain issues arose during the development of the programme which were time-consuming and demanding to resolve. They are briefed below.

* **Segmentation Fault**

Since there was abundant data to work with, one of the most frequently occurring errors was segmentation fault which often led to ‘core dumped’. It was realised after certain occurrences of this error that it is caused by the violation of memory access rights. Due to attempts to fetch data from memory locations which were not accessible, because they were not created or initialized with data, segmentation fault was often a problem that took time to resolve during the development of the programme.

* **Limitations of C Language**

Initially the programme was coded in C language. However, due to the limitations of this programming language, many of the features that the authors wished to implement could not be integrated efficiently. Furthermore, using C also led to great redundancy in the code and restricted the code to very basic and inefficient data structures. Most importantly, the manipulation of string literals or character arrays in C was getting increasingly difficult as well as redundant. Therefore, the program was revamped and written in C++ language which bears major resemblance to C but offers additional, powerful capabilities.

* **Extraction of data from the files**

Since there was only one file which contained the entire data the programme is supposed to deal with, it was a challenging task to extract individual records and the split them into different substrings to populate the arrays and linked lists. Moreover, another major problem with this was the use of open() and read() functions for file reading. Since the read() function only offers the option to read up to certain bytes, it was difficult to implement file reading using this function because every record varied in length so a certain length could not be specified which could cater to every single record for the URLs.

* **Implementation of nesting within priority queues**

While nesting within arrays can be easily implemented using for or while loops, it is slightly trickier to implement nesting in linked lists. This is why it was confusing for a while as to what approach would work the best and give results quickly for the data stored in the priority queues. Furthermore, it is important to mention that since linked lists are dynamic data structures which keep changing in size and data on the run, this factor also added to the complication of the process.

* **Implementation of the Graphical User Interface (GUI)**

While the actual implementation of the GUI in terms of its code and design was not difficult, it was the understanding of it that was very time-consuming. This was also due to the need to install multiple libraries, dependencies, and packages for GUI to run on Ubuntu 14.04. Apart from this, many warnings and errors occurred while implementing this which had to be understood in order to be resolved. The main issue with this was that there was a huge number of warnings which could be ignored but only one or two errors in between which needed to be found carefully.

Besides these major problems, there were multiple minor issues which arose including the use of ‘non-trivially copyable data type’.

**Limitations**

Despite efficiently providing the basic functionality of a web-crawler, our programme still has a limited scope at present. The following limitations are observed in the programme which can be fixed with modifications to the code.

* The programme is using hard-coded data which is placed in files. The data is being read from the files record by record (for each URL) and the different fields related to each URL are extracted from those records. However, the programme is not capable of fetching data from online sources or webpages which restricts its functionality to a very limited scope. Therefore, the programme can only process and extract information out of the data present in the local storage and not online resources.
* A single file holds all the data being used by the web-crawler. This limits the amount of data that can be searched by the user. The content related to each URL is placed at one place only which also makes traversing the file, for data searching, and extracting data from it, for array and linked list population, an arduous task. Furthermore, this makes data searching and extraction relatively slow.
* Another major limitation is concerned with the implementation of the Graphical User Interface (GUI). The GUI window appears only after the main programme exits. This means that the latest statistics would be displayed on the GUI and all other statistics for previous iterations of the searches will only be displayed on the Command Line Interface (CLI). Essentially, the GUI is unable to update data in real-time.
* Moreover, the programme is only able to cater to a string of words and extracts the data that exactly matches the input string. This implies that data which is relevant to the input but does not exactly match it character by character would not be displayed. As a result, the searches are limited to the precise string only rather than additional pertinent information as well.
* Furthermore, multi-threading is used to process the data of different Uniform Resource Locator (URLs) in groups. This has been achieved by creating four threads each of which processes the data of two URLs at a single instance. However, multi-programming has not been employed program-wide in the sense that only the fetching, extracting, and processing of the data is been done through multi-threading while the results are generated sequentially for each URL instead of in parallel with each other.

**Future Work**

There are numerous features which could be integrated into our programme to enhance its functionality and enable it to cater to more extensive data. Due of lack of time, the developers were unable to incorporate them. However, few of the ideas are listed below.

* The scope of the programme can be increased tremendously by integrating the feature of fetching data from online resources. This can be done easily with the use of certain Application Programming Interfaces (APIs). This will allow the searching of real-time data by the user. Interestingly, since the programme is already based on linked lists, there is virtually no limit to the data that can be fetched and processed by the programme, provided the local memory can cater to it.
* The data for each URL can be placed in a separate file. This will achieve two advantages. First of all, it will make fetching and searching for data extremely easy. Secondly, it will enable extensive content to be placed inside each file as opposed to limited content for all the URLs in a single file like the programme currently does.
* Moreover, the programme can be enhanced to enable the GUI to update in real-time and show statistics for every search instead of after the last one only. This can be implemented by tampering with the code for GUI in a way where it appears repeatedly while the programme is still going on.
* In addition to this, by employing more conditional statements and additional logic, the programme can be improved to display relevant data only instead of exactly matching data. For instance, if a person searches for ‘apples’, the programme currently only displays the webpages containing the word ‘apples’. However, after this improvement, it will be able to display those webpages as well which have information relevant to ‘apples’ even if they do not contain this particular keyword. That being said, it will become essential to carefully maintain the priority of the webpages in the priority-based queue since first those webpages should appear which contain this word, followed by the webpages containing relevant information but not this word.
* Lastly, it will largely enhance the efficiency and request response time of the programme if multi-threading can be employed programme-wide. The entire functionality that the programme provides, apart from the compilation and display of the results, should be handled via multiple threads. At present, a certain part of the programme is being executed under multi-threading. Updating its scope would make the programme more time-efficient and reduce the resources being utilised. However, the overhead due to the creation of threads needs to be considered before employing this feature.
* Apart from these, further nesting within the priority queues can allow the programme to trace data with finer detail and improve the speed of every traversal as well. Moreover, it will become easier to fetch data through random access method.

There are multiple other advanced features that can be added to improve the functionality, display, speed, and performance, etc. of the programme.

References

[1] N. Singhal, A. Dixit, and A.K. Sharma, Design of a Priority Based Frequency RegulatedIncremental Crawler, International Journal of Computer Applications, 1(1), 2010,.

[2] Ah Chung Tsoi, Daniele Forsali, Marco Gori, Markus Hagenbuchner, and Franco Scarselli, ASimple Focused Crawler, WWW2003 ACM, 2003.

[3] S.Ganesh, M.Jayaraj, V.Kalyan, S. Murthy, and G.Aghila, Ontology-based Web Crawler, Proc.IEEE International Conf. on Information Technology: Coding and Computing (ITCC’04),2004.

[4] D. Mukhopadhyay, A. Biswas, and S. Sinha, A New Approach to Design Domain SpecificOntology Based Web Crawler, Proc. 10th IEEE International Conf. on InformationTechnology, 2010.

[5] X. Chen, and X. Zhang, HAWK: A Focused Crawler with Content and Link Analysis, Proc.IEEE International Conf. on e-Business Engineering, 2008.

[6] D. Hati, B. Sahoo, and A Kumar, Adaptive Focused Crawling Based on Link Analysis, Proc.2nd IEEE International Conf. on Education Technology and Computer (ICETC), 2010.

[7] S.Thenmalar, and T. V. Geetha, Concept based Focused Crawling using Ontology,International Journal of Computer Applications, 26(7), 2011, 29-32.