DATABASE MANAGEMENT SYSTEMS
SUBMITTED TO : DR. WENDY OSBORN

IMAGE INDEXING WEB CRAWLER

Submission By : Michael Wilson & Taylor Kozar

A web crawler is a program or automated script which browses the World Wide Web in methodical, automated manner ([definition via] Science Daily, 2016). Many websites actually use web crawling as a means of searching out information from other sources and providing up to date information. In fact, one of the most major web crawlers, most of us use every day. Google uses a web crawling algorithm in order to find information pertaining to what people are searching for at any given moment. What is amazing about google however is millions of users all utilize it all at once.

In reality web crawlers are an essential component in our lives and they are important in data mining techniques as well as indexing for other applications. (Edwards, J., McCurley, K. S., and Tomlin, J. A., 2001) How they work is they start with URL's called seeds. The seeds are a starting point to search out further down into the web. The links stored to be searched out are known as the crawl frontier (Masanès, Julien, 2007) and these URLs are visited recursively, and if the crawler is set to do so it archives information off of the websites. The one major issue with archiving however is that it has to prioritize its archiving as if it visits millions of links it can't store all the information, for obvious reasons.

Since the size of the web is so large, there is now a new a method of selection implemented in every web crawler. This means that there is either a cap to the depth the search will actually go, or they index only a certain fraction of websites, this number usually between 40 and 70% (A. Gulli, A. Signorini, 2005). This is why being more specific in google searches will create more specific results, because, for example, if you search "web crawler" on google you'll likely get 40% of the relevant information pertaining to a web crawler and its information. On the other hand if you search "web crawler in modern day use" you receive data relevant to more recent studies rather than dating back into the era of the first web browsers.

Web crawlers are also used in a lot of modern day hacking. It is utilized because it accesses data that might not normally be available to the public, but can be indexed. An example of this is Facebook. If you go to the main Facebook page it makes you create an account in order to search for people. This is unavoidable, but there is a way around it if you use a web search for a specific person and have the word Facebook after it. The web crawler can seek out the pages with those extensions. So this is used in hacking because a website that may not be accessible to the public if accessing the main page, might be able to be accessed if I it can be backdoored so to speak, through a link that is directly accessed via the extensive link rather than a click link on the main page that redirects you.

Indexing images on the other hand can serve as a means of file sorting and a way of differentiating images to avoid duplications. Often on websites, databases and many computers images are duplicated and referenced from two different sources. Image indexing makes it so that multiple sources can all pull the same image from the same location.

By indexing images using pixel values as well you can make sure that values never equal the same thing because if the image is different it will have different pixel values, because that is just how images are made.

Content Based Image Retrieval, is a form of indexing for images that utilizes color patterns in order to retrieve images (Priyanka Malode, S. V. Gumaste, 2015) and this is can even be done on multiple dimensions of indexing. This meaning that multiple levels of images can be stored in order to accommodate huge libraries of images. This may be helpful when pertaining to photographers, graphic artists and web developers, because it can be used as a means of organizing images and having them be able to be quickly and easily requested and retrieved.

The project we implemented is a web crawler based program that grabs both web site links and images from websites that it accesses. Web crawling is a concept where an automated program, when given a starting web page, will parse through the source code of the given website and find other links in the web page. The crawler then accesses each of the websites individually and continues to dive deeper through the web using each of the new visited links and creating a figurative tree of these links. The tree stores the websites that have already been accessed thus when we try and access a website that has already been seen, hence if websites have multiple links to the same website, we don't check the websites again. This way we refrain from visiting the links more than once if they appear again, and it is a means to an end with the program, so that the web crawler does not find information not pertaining to what we are looking for.

The second part of the program relates to images. In addition to storing every web link it visits, we also store every image viewed in the website. We download the image to a file and each of these images can be viewed upon download. The images are also indexed into a file with a hashing function. This way we don’t get duplicated images and the file names are shortened and unique, so they aren’t full web page link names. These images can then be viewed upon request and are also viewable as the program runs in order to depict duplicated images and to show when new images have been added to the database. The image indexing is done by reading pixel values, and then averaging their RGB values. Essentially the program can provided with a ‘cut’ value and it divides the pictures in both directions by that number. It then averages the RGB values contained within each pixel of each section and uses this average to form part of the key. It repeats this process until the whole image has been evaluated and renames the files as the full hash value. This way if we retrieve to of the same images, they will have the same pixel values and when we index them they will be the same hash that is in our database already and we can refrain from storing it a second time.

What we use in order to implement the software is using a library known as libcurl. The library provides a basis for web crawling and downloading of files. It provided us with library support in order to actually retrieve images from the web and provided us functions to parse through the html code and find what we were looking for. It also provided a way of downloading the images to a specific folder where we could utilize and view them.

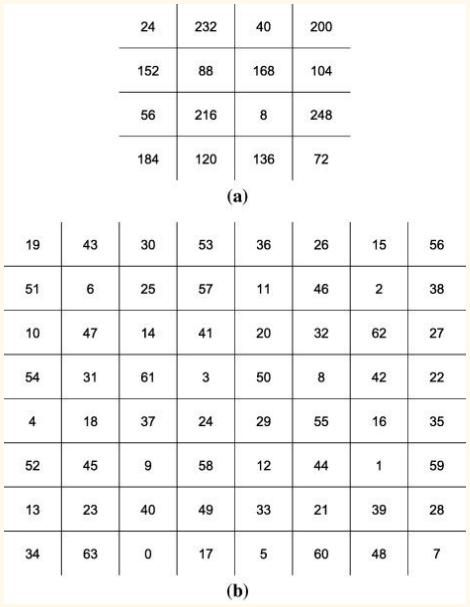
The libcurl library as defined by the provider is a client-side URL transfer library (libcurl index, 2016) that supports almost every web based extension and best of all it is open source so it is always improving and growing. This library allowed us to parse through any html file we provided the program and find any new links the website had. It then used these links to find more links and so on. Also the libcurl library allowed us to download the images to the computer in order to view them, and allowed the following library OpenCV to access them.

The next substantial part of the project was utilizing the library OpenCV. OpenCV, or Open Source Computer Vision (OpenCV, 2016) is a library that allows for image manipulation, image reading and pixel value analysis. It was the library used for the bulk of the project due to its ability to read in pixel values in which we could derive a hashing function for.

The idea was to have the images stored in two different ways. The first of which would be to download the images directly every time they are accessed and source them based on the pixels of the image. This would mean reading pixel values of the images and applying a hashing algorithm in order to store the images more efficiently and to avoid repetition. The second implementation we decided to do was to simply reference the links of all the images and have them be viewed in the web browser. This way we can save on computer space because none of the images need to be actually downloaded, and this way we can hash the pictures and store the key and then store the image location rather than the image itself.

The goal of this was to see which would be most efficient solely because there is a tradeoff between how much memory the images have to use and the time it takes to download all of them, and the amount of time it takes to store links and view each of them on the web page. This difference will tell us if web based imaging is quicker or if having the physical images on disk will be quicker.

The paper we used to as reference to implement the program talks about Ordered-Dither Block Truncation Coding. Ordered-Dither Block Truncation Coding is a form of image color compression. (Jing-Ming Guo, 2013) It does this by separating the pixel values of each of the values for RG and B and separates them into separate images so to speak.



24	232	40	200
152	88	168	104
56	216	8	248
184	120	136	72

(a)

19	43	30	53	36	26	15	56
51	6	25	57	11	46	2	38
10	47	14	41	20	32	62	27
54	31	61	3	50	8	42	22
4	18	37	24	29	55	16	35
52	45	9	58	12	44	1	59
13	23	40	49	33	21	39	28
34	63	0	17	5	60	48	7

(b)

Image via : (Jing-Ming Guo, 2013)

With this the it creates a bitmap image that is cut up into sections that are all separated and can be used to average out pixel values.

The way the paper does this is that it creates a Max Quantizer and Min Quantizer that store the minimum and maximum values for each of the three main color elements (Jing-Ming Guo, 2013), and combines them. From this combined color, an index is created and a color co-occurrence feature is derived from the color codebook implemented.

The way we covered this is we used bit mapping in order to separate the image into a bunch of sections. We then average the pixel values for each of the RGB values and implement a hash value using each of these. Therefore we get a unique key for every image and any images that are the same will be hashed to the same key and will actually be deleted before they are indexed. The one thing we did not implement was a color codebook. The reason for this was that it was rather unnecessary for deriving an index solely because a codebook in OpenCV is used for storing each of the frames of the image and the book can adjust the amount of time each section spends being processed, as well as tell if the an element has been changed. So essentially this would be more for image manipulation rather than just reading because a codebook would be necessary for making sure changes are properly stored.

The paper does not include a reference from images however. The images are simply provided to the program via a file and are done one by one as provided. Our program on the other hand indexes every image that the crawler retrieves. Meaning that the images are indexed as they come in. This way they are organized and the program execution is sequential and done in recursive calls until the web crawler is done searching. Once images are indexed, they are placed in a file that sorts the images based on their average pixel values.

In conclusion, the project was an overall success. Everything worked quite well and the program worked fast and efficiently. It was structurally similar to the paper we used as reference for the program and we are certainly happy with the turnout.

We learned that one thing the program failed to do properly in some cases was images that were png objects with only black text. The reason for this was that when OpenCV loads in a png image that doesn't have a background, it reads the pixel values as black. Therefore, when it hashes the images the hash values all end up being 0 which is the value for black and therefore a few images end up being the same if the images, even if the text inside was different. This is a very unlikely issue to happen, however it can happen in some cases, but this is a rather unavoidable situation.

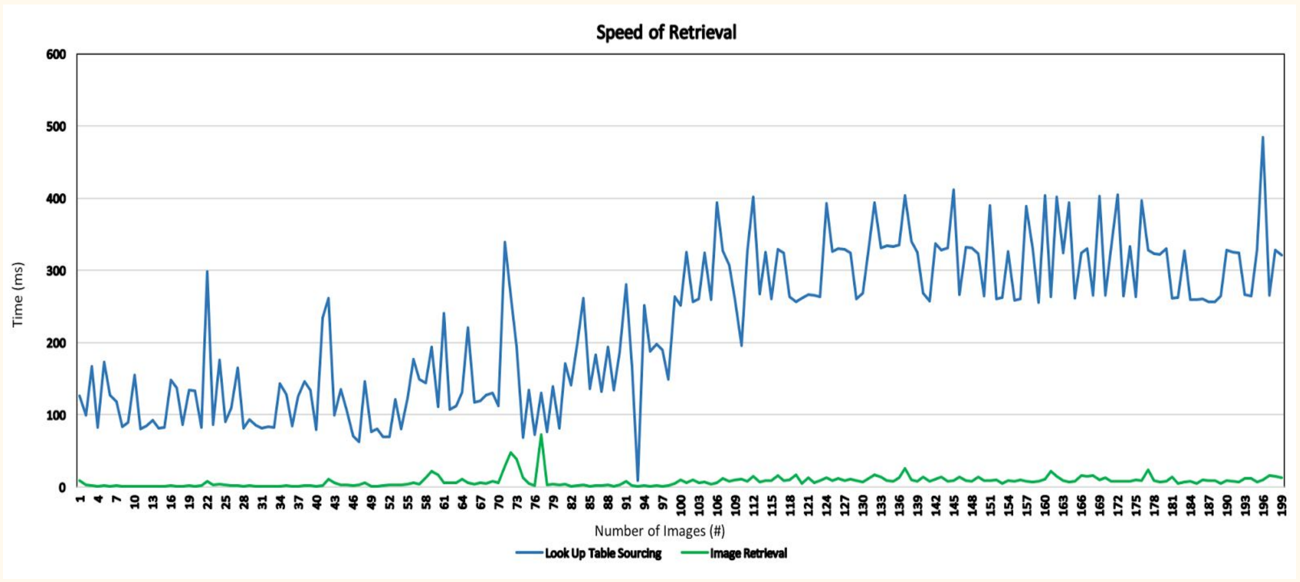
Another thing that seemed to have partial issues, but seems to have been resolved overall was if a link was verification based, it can get stuck in a loop trying to figure out if the link is accessible or not. There was no way of really checking for this other than simply timing out, but the problem is that the program wouldn't wait at the link it would simply find itself in an infinite loop causing the program to loop an verification check based website over and over. This issue was normally only for when the depth of the search neared the 10 mark, however it was still something to note during the process. This seems to be a rather minuet problem however solely because the likelihood of those showing up is very slight, but websites that included the "I am not a robot" verification links failed to process properly, for obvious reasons.

The project also has a certain degree of accuracy. The reason for this is that multiple pictures can have the same pixel averages even if the images are not the same. For example a picture that is cut in 4 pieces that has a black pixel in each of the sections. This pixel can be moved throughout that area and the pixel averages will still be the same as long as the pixel resides within that image. Thus if an image is structurally similar to another, it may find it to be the same image even if the images aren't 100% the same. The way of resolving this is a greater division factor, which would result in a much larger key value, ie (n^2)^3 where n is number of splits because the image would have a length and a height and a requirement to store all three RGB values.

The last thing of note with the project was any websites that were account based were likely to either show up with nothing or they would fail as well because of the verification purposes. So if you try and access a website that requires an account to access, ie) spotify premium, for example it would either loop the account verification or it would come back with no images and keep going. Again this is a rather small issue, and again it is for obvious reasons that it would be doing this. In the future a possible keychain could be implemented so that if a user actually has the accounts they could store them in the keychain and whenever an account is required the program could grab the appropriate login info and sign in to the website.

This software certainly shows why verification checks and accounts are required for a lot of websites because the software can literally search hundreds of links in a matter of minutes and grab hundreds of images in the same retrospect. These checks serve as a block to programs like ours so that we cannot access them with a virtual machine, we have to physically jump through the hoops to get where we want to be, this making it more secure and assures that an actual user is utilizing the software and not a computer virtually wasting their server space.

The result of our two softwares showed that images on file that were actually downloaded to the computer had an access time of less than 1 ms - 2 ms per image whereas images where the URL was used the data set ranged from 100 - 600 ms typically. This shows that it took over 100 times longer to get the images when they were not written on disc. The dataset is represented by the following graph :



This data was retrieved from our software where we store the access time to a text file for each image that the program retrieves. While graphed we can see a significant improvement in the speed of access when the image is already on disk. Overall the images, if they remain under about 800, stay at about 100 MB of memory usage roughly, while storing the website links uses about 22 KB. So if storage is a big issue then the look-up table is significantly better on storage use however if speed is the issue then the access time of the images on disc is exponentially better when the images start to get higher and higher in resolution.

Sources :

Cosenza, Mimmo (December 30th, 2014). Introduction to OpenCV Development with Clojure, Vol : 1, Version : 3.0.0

http://docs.opencv.org/3.0-last-rst/doc/tutorials/introduction/clojure_dev_intro/clojure_dev_intro.html

Daily, Science (2016), Web Crawler.
https://www.sciencedaily.com/terms/web_crawler.htm

Edwards, J., McCurley, K. S., & Tomlin, J.A. (2001). An adaptive model for optimizing performance of an incremental web crawler, Vol : 10, 106-113

Gully, A., Signorini, A. (2005). The Indexable Web Is More Than 11.5 Billion Pages, Vol : 14, 902-903

Guo, Jing-Ming (November 2013). Journal of visual communication and image representation, Vol : 24, Issue: 8, 1360-1379.

Malode, Priyanka & Gumaste, S. V. (December 2015). A Review Paper on Content Based Image Retrieval, Vol : 2, Issue : 9, 883-885.
https://www.irjet.net/archives/V2/i9/IRJET-V2I9149.pdf

Masanes, Julien (February 15, 2007). Web Archiving. Edition : 1, Page 132-137
Accessed via ebook publication.

Open Source, Libcurl (2016)
https://curl.haxx.se/libcurl/