

PROJECT REPORT
ON
Crawler Based Search Engine
Submitted in Partial Fulfillment of the
Requirements for the Award of Degree
of
BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE & ENGINEERING

SUBMITTED BY
BHUPENDER(2514211)
KULDEEP(2514217)
SUBMITTED TO
Dr. KARAMBIR
(ASSISTANT PROFESSOR,DEPARTMENT OF COMPUTER SCIENCE)



COMPUTER SCIENCE & ENGINEERING
UNIVERSITY INSTITUTE OF ENGINEERING AND TECHNOLOGY,
KURUKSHETRA UNIVERSITY, KURUKSHETRA-136119

2014-2018

ACKNOWLEDGEMENT

First and foremost we would like to express our gratitude to our guide **Dr. Karambir** and other faculty member for giving us wonderful opportunity to work on the project. We are very thankful to him who was always ready to lend their helping hand to us. This project has giving us fair exposure to some of the very interesting features of “**Crawler Based Search Engine**”.

We are also thankful to all my teachers of UNIVERSITY INSTITUTE OF ENGINEERING & TECHNOLOGY, KURUKSHETRA UNIVERSITY, KURUKSHETRA who were simply full ideas and whenever there was any need, they shared those great ideas and concept with us. And in the end, we would like to thank all those who helped us during the testing phase of the project.

DECLARATION

We hereby declare that the work which is being presented in the project, entitled “**Crawler Based Search Engine**” for the award of the degree of BACHELOR of Technology in Computer Science and Engineering is an authentic record of our own work carried out under the supervision of Dr. Karambir, Assistant Professor, Department of C.S.E., U.I.E.T, Kurukshetra University, Kurukshetra. The matter presented in this project has not been submitted by us for the award of any degree of this or any other University.

Further, we declare that where other’s ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

November,2017

Bhupender & Kuldeep

CERTIFICATE

This is to certify that the project entitled “**Crawler Based Search Engine**” submitted by **Bhupender, Roll No. 2514211 & Kuldeep, Roll No. 2514217** to the Department of CSE of U.I.E.T, Kurukshetra University, Kurukshetra for the award of the Degree of Bachelor of Technology in CSE, is a project work carried out by them under my supervision and guidance. Their project has reached the standard of fulfilling the requirements of regulations relating to degree.

I wish them success in all their future endeavors.

This is to certify that Bhupender, Roll No. 2514211 & Kuldeep, Roll No. 2514217 are bonafide students of Bachelor of Technology in CSE. The dissertation is, in our opinion worthy for consideration for the award of Bachelor of Technology in CSE.

Bhupender & Kuldeep

B.Tech.

Dept. of CSE.

U.I.E.T, KUK

Dr. Karambir

Assistant Professor

Dept. of CSE.

U.I.E.T, KUK

CHAPTER-1

1. INTRODUCTION:

1.1 INTRODUCTION TO PROJECT:

I have built a simple search engine using python. This search engine, returns the list of url's that contain a specific keyword in the pages the engine had crawled till then. So, in the main function, we first let the function crawl, giving a specific url as seed, forming an index. Then we go for looking the word entered to search in the index (We can search for any keyword). The Look_up function helps us in our later task returning a list of urls that contain our keyword.

The ability to search a specific web site for the page you are looking for is a very useful feature. However, searching can be complicated and providing a good search experience can require knowledge of multiple programming languages. This project will demonstrate a simple search engine you can run in your own site. This project is also a good introduction to the Python programming language. You can run this project on any server which supports CGI and has Python installed. This project was tested with Python version 2.7.

Reasons to choose Python:

There are a lot of web scripting languages and tools available. PERL and Ruby come quickly to mind, but there are many more. Python is a dynamically typed object oriented language. In comparison to Java, Python allows you to reassign object types. Python does not require all code to be within an object in the way that Java does. Python can also work more like a traditional scripting language with less object use.

PERL has a specialized syntax which can be difficult to learn and Ruby most commonly relies on the RAILS framework. They are both very popular and this application could have easily been written with either of them. This project could have been written in any one of those languages. Python just happens to be the language I was most interested in when I wrote this code.

Features:

- Anytime and anywhere access
- Crawling in all web pages
- Ranking the crawled pages
- Finding and printing the index

Advantages:

The following steps that give the detailed information of the need of proposed system are:

Performance: During past several decades, the records are supposed to be manually handled for all activities. The manual handling of the record is time consuming and highly prone to error. To improve the performance of the Hotel Management System, the computerized system is to be undertaken. This project is fully computerized and user friendly even that any of the members can see the report and status of the company.

Efficiency: The basic need of this website is efficiency. The website should be efficient so that whenever a new user submits his/her details the website is updated automatically. This record will be useful for other users instantly.

Control: The complete control of the project is under the hands of authorized person who has the password to access this project and illegal access is not supposed to deal with. All the control is under the administrator and the other members have the rights to just see the records not to change any transaction or entry.

Security: Security is the main criteria for the proposed system. Since illegal access may corrupt the database. So security has to be given in this project.

1.1.1 TOOLS AND LANGUAGES USED:

- Python
- IDLE(python gui)
- Python(command line)

Python:

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). This tutorial gives enough understanding on Python programming language.

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- Python is Interpreted – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive – You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

- Python is Object-Oriented – Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- Python is a Beginner's Language – Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Python Features:

Python's features include –

- Easy-to-learn – Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- Easy-to-read – Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain – Python's source code is fairly easy-to-maintain.
- A broad standard library – Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Interactive Mode – Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- Portable – Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable – You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases – Python provides interfaces to all major commercial databases.
- GUI Programming – Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- Scalable – Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below –

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

1.1.2 TYPES OF SEARCH ENGINES:

Although the term "search engine" is often used indiscriminately to describe crawlerbased search engines, human-powered directories, and everything in between, they are not all the same. Each type of "search engine" gathers and ranks listings in radically different ways.

1. Crawler-Based

Crawler-based search engines such as Google, compile their listings automatically. They "crawl" or "spider" the web, and people search through their listings. These listings are what make up the search engine's index or catalog. You can think of the index as a massive electronic filing cabinet containing a copy of every web page the spider finds. Because spiders scour the web on a regular basis, any changes you make to a web site may affect your search engine ranking. It is also important to remember that it may take a while for a spidered page to be added to the index. Until that happens, it is not available to those searching with the search engine.

2.Directories

Directories such as Open Directory depend on human editors to compile their listings. Webmasters submit an address, title, and a brief description of their site, and then editors review the submission. Unless you sign up for a paid inclusion program, it may take months for your web site to be reviewed. Even then, there's no guarantee that your web site will be accepted. After a web site makes it into a directory however, it is generally very difficult to change its search engine ranking. So before you submit to a directory, spend some time working on your titles and descriptions. Moreover, make sure your pages have solid well-written content.

3.Mixed Results /hybrid search engine

Some search engines offer both crawler-based results and human compiled listings. These hybrid search engines will typically favor one type of listing over the other however. Yahoo for example, usually displays human-powered listings. However, since it draws secondary results from Google, it may also display crawler-based results for more obscure queries. Many search engines today combine a spider engine with a directory service. The directory normally contains pages that have already been reviewed and accessed.

4.Pay Per Click

More recently, search engines have been offering very cost effective programs to ensure that your ads appear when a visitor enters in one of your keywords. This new trend is to charge you on a Cost per Click model (CPC).The listings are comprised entirely of advertisers who have paid to be there. With services such as Yahoo SM, Google AdWords, and FindWhat, bids determine search engine ranking. To get top ranking, an advertiser just has to outbid the competition.

5.Metacrawlers Or Meta Search Engines

Metasearch Engines search, accumulate and screen the results of multiple Primary Search Engines (i.e. they are search engines that search search engines) Unlike search engines,

metacrawlers don't crawl the web themselves to build listings. Instead, they allow searches to be sent to several search engines all at once. The results are then blended together onto one page.

The search engine can be categorized as follows based on the application for which they are used:

1.Primary Search Engines:

They scan entire sections of the World Wide Web and produce their results from databases of Web page content, automatically created by computers.

2.Subject Guides :

They are like indexes in the back of a book. They involve human intervention in selecting and organizing resources, so they cover fewer resources and topics but provide more focus and guidance

3.People Search Engines :

They search for names, addresses, telephone numbers and e mail address.

4.Business and Services Search Engines:

They essentially national yellow page directories .

5.Employment and Job Search Engines:

They either (a) provide potential employers access to resumes of people interested in working for them or (b) provides prospective employees with information on job availability .

6.Finance-Oriented Search Engines:

They facilitate searches for specific information about companies(officers, annual reports, SEC filings, etc.)

7.News Search Engines :

They search newspaper and news web site archives for the selected information

8.Image Search Engines:

They which help you search the WWW for images of all kinds.

9.Specialized Search Engines:

They that will search specialized databases, allow you to enter your search terms in a particularly easy way, look for low prices on items you are interested in purchasing, and even give you access to real, live human beings to answer your questions

1.1.3 WORKING OF SEARCH ENGINE

Search engines use automated software programs known as spiders or bots to survey the Web and build their databases. Web documents are retrieved by these programs and analyzed. Data

collected from each web page are then added to the search engine index. When you enter a query at a search engine site, your input is checked against the search engine's index of all the web pages it has analyzed. The best URLs are then returned to you as hits, ranked in order with the best results at the top.

The working of a search engine can be thus divided into three steps

1. Web Crawling
2. Indexing
3. Ranking

1. WEB CRAWLING

A web crawler (also known as a web spider or web robot) is a program or automated script which browses the World Wide Web in a methodical, automated manner. Other less frequently used names for web crawlers are ants, automatic indexers, bots, and worms. This process is called web crawling or spidering. Many legitimate sites, in particular search engines, use spidering as a means of providing up-to-date data. Web crawlers are mainly used to create a copy of all the visited pages for later processing by a search engine, that will index the downloaded pages to provide fast searches. Crawlers can also be used for automating maintenance tasks on a web site, such as checking links or validating HTML code. Also, crawlers can be used to gather specific types of information from Web pages, such as harvesting e-mail addresses (usually for spam).

A web crawler is one type of bot, or software agent. In general, it starts with a list of URLs to visit, called the seeds. As the crawler visits these URLs, it identifies all the hyperlinks in the page and adds them to the list of URLs to visit, called the crawl frontier. URLs from the frontier are recursively visited according to a set of policies.

Crawling policies

There are two important characteristics of the Web that generate a scenario in which web crawling is very difficult: its large volume and its rate of change, as there are a huge number of pages being added, changed and removed every day. Also, network speed has improved less than current processing speeds and storage capacities. The large volume implies that the crawler can only download a fraction of the Web pages within a given time, so it needs to prioritize its downloads. The high rate of change implies that by the time the crawler is downloading the last pages from a site, it is very likely that new pages have been added to the site, or that pages have already been updated or even deleted. A crawler must carefully choose at each step which pages to visit next. The behavior of a web crawler is the outcome of a combination of policies:

- A selection policy that states which pages to download.
- A re-visit policy that states when to check for changes to the pages.

- A politeness policy that states how to avoid overloading websites.
- A parallelization policy that states how to coordinate distributed web crawlers.

Selection policy

Given the current size of the Web, even large search engines cover only a portion of the publicly available content; no search engine indexes more than 16% of the Web. As a crawler always downloads just a fraction of the Web pages, it is highly desirable that the downloaded fraction contains the most relevant pages, and not just a random sample of the Web.

This requires a metric of importance for prioritizing Web pages. The importance of a page is a function of its intrinsic quality, its popularity in terms of links or visits, and even of its URL (the latter is the case of vertical search engines restricted to a single top-level domain, or search engines restricted to a fixed Website). Designing a good selection policy has an added difficulty: it must work with partial information, as the complete set of Web pages is not known during crawling.

Various selection policies are:

A crawler may only want to seek out HTML pages and avoid all other MIME types. In order to request only HTML resources, a crawler may make an HTTP HEAD request to determine a web resource's MIME type before requesting the entire resource with a GET request. To avoid making numerous HEAD requests, a crawler may alternatively examine the URL and only request the resource if the URL ends with .html, .htm or a slash. This strategy may cause numerous HTML web resources to be unintentionally skipped.

Some crawlers may also avoid requesting any resources that have a "?" in them (are dynamically produced) in order to avoid spider traps which may cause the crawler to download an infinite number of URLs from a website.

1. Path-ascending crawling

Some crawlers intend to download as many resources as possible from a particular web site. Cothey (Cothey, 2004) introduced a path-ascending crawler that would ascend to every path in each URL that it intends to crawl. For example, when given a seed URL of <http://foo.org/a/b/page.html>, it will attempt to crawl /a/b/, /a/, and /. Cothey found that a path-ascending crawler was very effective in finding isolated resources, or resources for which no inbound link would have been found in regular crawling.

2. Focused crawling

The importance of a page for a crawler can also be expressed as a function of the similarity of a page to a given query. Web crawlers that attempt to download pages that are similar to each other are called focused crawlers or topical crawlers. Focused crawling was first introduced by Chakrabarti et al.

The main problem in focused crawling is that in the context of a web crawler, we would like to be able to predict the similarity of the text of a given page to the query before actually downloading the page. The performance of a focused crawling depends mostly on the richness of links in the specific topic being searched, and a focused crawling usually relies on a general Web search engine for providing starting points.

3. Crawling the Deep Web

A vast amount of web pages lie in the deep or invisible web. These pages are typically only accessible by submitting queries to a database, and regular crawlers are unable to find these pages if there are no links that point to them. Google's Sitemap Protocol and mod_oai (Nelson et al., 2005) are intended to allow discovery of these deep-web resources.

Re-visit policy

The Web has a very dynamic nature, and crawling a fraction of the Web can take a long time, usually measured in weeks or months. By the time a web crawler has finished its crawl, many events could have happened. These events can include creations, updates and Deletions. The objective of the crawler is to keep the average freshness of pages in its collection as high as possible, or to keep the average age of pages as low as possible. These objectives are not equivalent: in the first case, the crawler is just concerned with how many pages are out-dated, while in the second case, the crawler is concerned with how old the local copies of pages are. Two simple re-visiting policies were studied by Cho and Garcia-Molina

1. Uniform policy:

This involves re-visiting all pages in the collection with the same frequency, regardless of their rates of change.

2. Proportional policy:

This involves re-visiting more often the pages that change more frequently. The visiting frequency is directly proportional to the (estimated) change frequency. In terms of average freshness, the uniform policy outperforms the proportional policy in both a simulated Web and a real Web crawl. The explanation for this result comes from the fact that, when a page changes too often, the crawler will waste time by trying to re-crawl it too fast and still will not be able to keep its copy of the page fresh.

Politeness policy

Crawlers can retrieve data much quicker and in greater depth than human searchers, so they can have a crippling impact on the performance of a site. Needless to say if a single crawler is performing multiple requests per second and/or downloading large files, a server would have a hard time keeping up with requests from multiple crawlers.

As noted by Koster (Koster, 1995), the use of Web crawlers is useful for a number of tasks, but comes with a price for the general community. The costs of using Web crawlers include:

- Network resources, as crawlers require considerable bandwidth and operate with a high degree of parallelism during a long period of time.
- Server overload, especially if the frequency of accesses to a given server is too high.
- Poorly written crawlers, which can crash servers or routers, or which download pages they cannot handle. Personal crawlers that, if deployed by too many users, can disrupt networks and Web servers.

A partial solution to these problems is the robots exclusion protocol, also known as the robots.txt protocol (Koster, 1996) that is a standard for administrators to indicate which parts of their Web servers should not be accessed by crawlers. This standard does not include a suggestion for the interval of visits to the same server, even though this interval is the most effective way of

avoiding server overload. A non-standard robots.txt file may use a "Crawl-delay:" parameter to indicate the number of seconds to delay between requests, and some commercial search engines like MSN and Yahoo will adhere to this interval. Anecdotal evidence from access logs shows that access intervals from known crawlers vary between 20 seconds and 3–4 minutes. It is worth noticing that even when being very polite, and taking all the safeguards to avoid overloading Web servers, some complaints from Web server administrators are received.

Parallelization policy

Main article: Distributed web crawling

A parallel crawler is a crawler that runs multiple processes in parallel. The goal is to maximize the download rate while minimizing the overhead from parallelization and to avoid repeated downloads of the same page. To avoid downloading the same page more than once, the crawling system requires a policy for assigning the new URLs discovered during the crawling process, as the same URL can be found by two different crawling processes. Cho and Garcia-Molina (Cho and Garcia-Molina, 2002) studied two types of policies:

1. Dynamic assignment:

With this type of policy, a central server assigns new URLs to different crawlers dynamically. This allows the central server to, for instance, dynamically balance the load of each crawler. With dynamic assignment, typically the systems can also add or remove downloader processes. The central server may become the bottleneck, so most of the workload must be transferred to the distributed crawling processes for large crawls. There are two configurations of crawling architectures with dynamic assignments that have been described by Shkapenyuk and Suel (Shkapenyuk and Suel, 2002):

- A small crawler configuration, in which there is a central DNS resolver and central queues per website, and distributed downloaders.
- A large crawler configuration, in which the DNS resolver and the queues are also distributed.

2. Static assignment:

With this type of policy, there is a fixed rule stated from the beginning of the crawl that defines how to assign new URLs to the crawlers. For static assignment, a hashing function can be used to transform URLs (or, even better, complete website names) into a number that corresponds to the index of the corresponding crawling process. As there are external links that will go from a website assigned to one crawling process to a website assigned to a different crawling process, some exchange of URLs must occur.

To reduce the overhead due to the exchange of URLs between crawling processes, the exchange should be done in batch, several URLs at a time, and the most cited URLs in the collection should be known by all crawling processes before the crawl (e.g.: using data from a previous crawl) (Cho and Garcia-Molina, 2002).

An effective assignment function must have three main properties: each crawling process should get approximately the same number of hosts (balancing property), if the number of crawling processes grows, the number of hosts assigned to each process must shrink (contra-variance property), and the assignment must be able to add and remove crawling processes dynamically. Boldi et al. (Boldi et al., 2004) propose to use consistent hashing, which replicates the buckets, so adding or removing a bucket does not require re-hashing of the whole table to achieve all of

the desired properties. crawling is an effective process synchronisation tool between the users and the search engine.

URL normalization

Crawlers usually perform some type of URL normalization in order to avoid crawling the same resource more than once. The term URL normalization, also called URL canonicalization, refers to the process of modifying and standardizing a URL in a consistent manner. There are several types of normalization that may be performed including conversion of URLs to lowercase, removal of "." and ".." segments, and adding trailing slashes to the non-empty path component

Crawler identification

Web crawlers typically identify themselves to a web server by using the User-agent field of an HTTP request. Website administrators typically examine their web servers' log and use the user agent field to determine which crawlers have visited the web server and how often. The user agent field may include a URL where the website administrator may find out more information about the crawler. Spambots and other malicious web crawlers are unlikely to place identifying information in the user agent field, or they may mask their identity as a browser or other well-known crawler.

It is important for web crawlers to identify themselves so website administrators can contact the owner if needed. In some cases, crawlers may be accidentally trapped in a crawler trap or they may be overloading a web server with requests, and the owner needs to stop the crawler. Identification is also useful for administrators that are interested in knowing when they may expect their web pages to be indexed by a particular search engine.

2. INDEXING

The web pages searched by the Web Crawlers have to be indexed for future use so that whenever search for related articles is asked the result can be provided by looking up the index. The index stores the URL along with the keywords relevant to the page. It may also store information like the date on which the page was last visited and also the date on which the page should be revisited under the revisiting policy along with the age and freshness factors.

3. RANKING

Most of the search engines return results with confidence or relevancy rankings. In other words, they list the hits according to how closely they think the results match the query. However, these lists often leave users shaking their heads on confusion, since, to the user, the results may seem completely irrelevant.

Basically this happens because search engine technology has not yet reached the point where humans and computers understand each other well- enough to communicate clearly. Most search engines use search term frequency as a primary way of determining whether a document is relevant. If you're researching diabetes and the word "diabetes" appears multiple times in a Web document, it's reasonable to assume that the document will contain useful information.

Therefore, a document that repeats the word "diabetes" over and over is likely to turn up near the top of your list.

If your keyword is a common one, or if it has multiple other meanings, you could end up with a lot of irrelevant hits. And if your keyword is a subject about which you desire information, you don't need to see it repeated over and over--it's the information about that word that you're interested in, not the word itself.

Some search engines consider both the frequency and the positioning of keywords to determine relevancy, reasoning that if the keywords appear early in the document, or in the headers, this increases the likelihood that the document is on target. For example, one method is to rank hits according to how many times your keywords appear and in which fields they appear (i.e., in headers, titles or plain text). Another method is to determine which documents are most frequently linked to other documents on the Web. The reasoning here is that if other folks consider certain pages important, you should, too. If you use the advanced query form on AltaVista, you can assign relevance weights to your query terms before conducting a search. Although this takes some practice, it essentially allows you to have a stronger say in what results you will get back. As far as the user is concerned, relevancy ranking is critical, and becomes more so as the sheer volume of information on the Web grows. Most of us don't have the time to sift through scores of hits to determine which hyperlinks we should actually explore. The more clearly relevant the results are, the more we're likely to value the search engine.

4. STORAGE COSTS AND CRAWLING TIME

Storage costs are not the limiting resource in search engine implementation. Simply storing 10 billion pages of 10kbytes each (compressed) requires 100TB and another 100TB or so for indexes, giving a total hardware cost of under \$200k: 400 500GB disk drives on 100 cheap PCs. However, a public search engine requires considerably more resources than this to calculate query results and to provide high availability. And the costs of operating a large server farm are not trivial. Crawling 10B pages with 100 machines crawling at 100 pages/second would take 1M seconds, or 11.6 days on a very high capacity Internet connection. Most search engines crawl a small fraction of the web (10-20% pages) at around this frequency or better, but also crawl dynamic web sites (e.g. news sites and blogs) at a much higher frequency

1.2 OBJECTIVE:

A search engine or search service is a program designed to help find information stored on a computer system such as the World Wide Web, inside a corporate or proprietary network or a personal computer. The search engine allows one to ask for content meeting specific criteria (typically those containing a given word or phrase) and retrieves a list of references that match those criteria. Search engines use regularly updated indexes to operate quickly and efficiently.

Search Engine usually refers to a Web search engine, which searches for information on the public Web. However there are other kinds of search engines like enterprise search engines, which search on intranets, personal search engines, which search individual personal computers, and mobile search engines.

Some search engines also mine data available in newsgroups, large databases, or open directories like DMOZ.org. Unlike Web directories, which are maintained by human editors, search engines

operate algorithmically. Most web sites which call themselves search engines are actually front ends to search engines owned by other companies.

1.3 THE PROPOSED SYSTEM

This is a simple search engine, I implemented in Python. This is a Python console program. The inputs that you have to provide when it asks are as follows:

Seed Page - This is the page, from which it starts crawling the web. Give the web url of a good seed page, that has ample links on it, so that it can crawl into those pages and again crawl from those pages into other pages eg:<http://opencvuser.blogspot.in>

Maximum depth - This is the number of links to crawl completely. It would take 30 second for first link and for second link 60 seconds and it keeps on doubling. So, maximum 10 links are more than enough, I would say. eg:10

After you give the above inputs, the program starts running. It may take a lot of time, to crawl depending on the depth you have specified. So the depth number, will be visible, decrementing itself, when ever a link is completely crawled; so that when it reaches 0 you know the crawling ended.

1. Web Crawler :

We will be using Python to build the search engine.

If you look at a web page, it contains a lot of information. There will be pictures, videos, buttons and what not. If we try to crawl them directly, it's a huge tedious task. What we will be crawling is the source of the web document. The easiest way to view this is Rightclick-> ViewSource or View Page Source on the webpage you are viewing. Actually, this is the only information, that a Web server sends when a browser requests the page. What we actually see is it's rendering by our Web Browser.

Now, our aim is to find all the links in a given page source of a website or more specifically from a single webpage. How the page source is "Given" will be a future topic and will be covered in latter posts. Also, let us first write, how to extract the first link that appears in that source page, and then in the next part of this topic, we will extract all the links.

In the variable a, we are storing a sample html source with two links. Links may be in many ways, but we are assuming that they will start in the most standard manner with <a href> tags.

Inorder to extract the first link from a , we are exploring the a using the string find method of Python. This method gives the first occurrence of the substring we are searching. The location is the offset of the substring from the beginning of the string. It's first argument is the string to be

searched for and the second optional one, from which location to start searching. So, in start , we are storing the location where the a of a href occurs. After that, we need to find closing quotes of the URL to know where it ends. The starting quotes of the URL occur at location start+7 , so from there on, we can search for the End of the URL where the quotes end. Then we are printing the URL.

What this program does, it will be better to explain in a step by step manner.

1. A url of the seed page is taken and passed to the Crawl_web function
2. The Crawl_web function takes the passed url as input string. It crawl the entire webpage with the given url and finds all the url's with <a href> tag in the page
3. Then it takes one of those url's, sees that it was not already crawled and goes to step-1. If all the url's are over, then it goes to the next step
4. The Crawl_web function returns the list of all the url's obtained by crawling starting from the given seed page
5. We print the url's

2. Indexing

The web pages searched by the Web Crawlers have to be indexed for future use so that whenever search for related articles is asked the result can be provided by looking up the index. The index stores the URL along with the keywords relevant to the page. It may also store information like the date on which the page was last visited and also the date on which the page should be revisited under the revisiting policy along with the age and freshness factors.

3. Ranking

Most of the search engines return results with confidence or relevancy rankings. In other words, they list the hits according to how closely they think the results match the query. However, these lists often leave users shaking their heads on confusion, since, to the user, the results may seem completely irrelevant. Basically this happens because search engine technology has not yet reached the point where humans and computers understand each other well enough to communicate clearly.

Most search engines use search term frequency as a primary way of determining whether a document is relevant. If you're researching diabetes and the word "diabetes" appears multiple times in a Web document, it's reasonable to assume that the document will contain useful information.

Therefore, a document that repeats the word "diabetes" over and over is likely to turn up near the top of your list. If your keyword is a common one, or if it has multiple other meanings, you could

end up with a lot of irrelevant hits. And if your keyword is a subject about which you desire information, you don't need to see it repeated over and over--it's the information about that word that you're interested in, not the word itself. Some search engines consider both the frequency and the positioning of keywords to determine relevancy, reasoning that if the keywords appear early in the document, or in the headers, this increases the likelihood that the document is on target. For example, one method is to rank hits according to how many times your keywords appear and in which fields they appear (i.e., in headers, titles or plain text). Another method is to determine which documents are most frequently linked to other documents on the Web. The reasoning here is that if other folks consider certain pages important, you should, too.

If you use the advanced query form on AltaVista, you can assign relevance weights to your query terms before conducting a search. Although this takes some practice, it essentially allows you to have a stronger say in what results you will get back. As far as the user is concerned, relevancy ranking is critical, and becomes more so as the sheer volume of information on the Web grows.

Most of us don't have the time to sift through scores of hits to determine which hyperlinks we should actually explore. The more clearly relevant the results are, the more we're likely to value the search engine.

CHAPTER-2

2. REQUIREMENT SPECIFICATION

2.1 HARDWARE AND SOFTWARE REQUIREMENTS

Hardware requirements:

1. System processor – intel i3 minimum
2. Mobile processor – 1GHz or higher
3. RAM – 4GB minimum

Software requirements:

1. Operating system(Windows 7, Windows 8, Windows 10)
2. Browser (Google Chrome, Internet Explorer)

2.2 FUNCTIONAL REQUIREMENTS

Function requirements mean the physical modules, which are going to be produced in the proposed systems. The functional requirements of the proposed system are described below:

- **Web crawling:**

The system will provide web crawler. It can create an index of available City University web pages. It takes the index page of City University, i.e. www.cityu.edu.hk, as a starting point. After it starts, it gets texts (contents) from each web page it reaches and also extracts the URLs of the hyperlinks in each page and puts them into queue. This queue provides the web crawler with the URLs that it has to visit.

- **Words Extracting:**

The system will provide string tokenizer. After web crawler has indexed the pages. The texts of each page have to be extracted from that page, if not, they are just rubbish mixing with HTML. String tokenizer method is used to extract content words from the HTML, ASP, or etc. pages.

- **Ranking :**

The system will provide a rank calculating function. The rank of a word is based on prominence and frequency, as explained before.

- **Search Results Ordering:**

The search results from search function are not in order. This function can reorder the results by ranking number in descending order, i.e. the most relevant results are ordered at the most beginning

2.3 SOFTWARE DEVELOPMENT

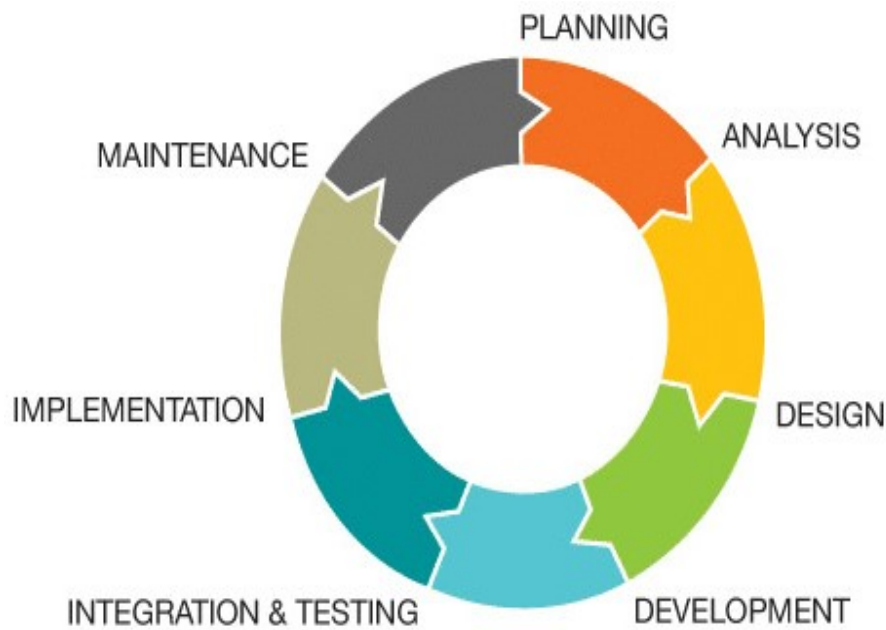


Figure 2.1

2.3.1 FEASIBILITY STUDY

Feasibility study is carried out to test if the proposed system is worth being implemented. Given unlimited resources and infinite time, all projects are feasible. Unfortunately, such situations are not possible in real time. Hence it becomes necessary and prudent to evaluate the feasibility of the project at earliest possible time in order to avoid unnecessary wastage of time.

Feasibility study is the test of the system proposed regarding its work ability, impact or organization's ability to meet user's needs and effective use of resources. It is usually carried out by a small group of people who are familiar with the information system techniques, understand the part of business that will be involved and affected by the people that are skilled in analysis and design.

A feasibility study is conducted to select the best system that meets the performance requirements. This entails an identification description, and emulation of candidate systems and selection of best system for the job.

The factors that should be included in the feasibility assessment can be as follows:

- Cost: operating, maintenance, unit
- Accuracy: frequency, significance and correction of errors
- Reliability: stability, durability
- Capacity: average, low and peak loads

Economic Feasibility

Economic analysis is the most frequently used method for evaluating the effectiveness of a candidate system. It is more commonly known as cost/benefits and saving that are expected from a candidate system and compare them with cost. If benefits outweigh costs, then the decision is made to design and implement the system.

Usually cost benefits analysis is made to find the savings or extra overheads that would arise new development. The technique of cost benefit analysis is often used as a basis for assessing economic feasibility. The factors for evaluation are:

- Cost of operation of existing system and proposed system
- Cost of development of proposed system
- Value of benefits of proposed system

Technical Feasibility

Technical feasibility centers on existing computer system and to what extent it can support the proposed addition. This involves financial consideration to accumulate technical enhancement. E.g. if the current operating system is at 80% capacity and arbitrary ceiling then running another application could overload the system or require additional hardware. If the budget is serious constraint then the project is not feasible.

Operational Feasibility

The operational feasibility refers to the assessment of proposed system in the manner that how much this system is feasible for the end users. The system should have capabilities in it. That person with a simple knowledge can also use the system. Our proposed system is user-friendly interface. The users just have to click on the choice with the help of menu. Therefore the system is feasible on operational front too.

Our system will improve the performance and save the time. Because of the simple interface user can easily navigate to the desired information page and hence can get the desired information.

Time feasibility

Time feasibility determines whether system is implemented within stipulated time. This project will be completed within stipulated time frame.

2.3.2 REQUIREMENT SPECIFICATIONS

Requirement analysis is a software engineering task that bridges the gap between system level software analysis and software design.

Requirement analysis enables the system engineer to specify software function and performance indicate s/w interface with other system elements and establish constraints that software must

meet. Requirements analysis allows the software engineer to refine the software allocation and build modules of the data, function and behavior domain that will be treated by software. Requirement specification provides the description to the developer and the customer with the mean to access quality rule.

There are four basic elements in system requirements analysis:

- Output

First of all, we must determine what the objectives or goals are, what do we intend to achieve, what is the purpose of our work; in other words what is the main aim behind the system. Defining aim is very vital in system work. If we do not know where we want to go, we will not know when we have reached there; we shall be unnecessarily wasting our time and energy in the system. The user department has to define these objectives in terms of their needs. These become the output, which the system analyst keeps into mind.

- Input

Once we know the output, we can easily determine when the inputs should be sometimes, it may happen that the required information may not be readily available in the proper form. This may be because of the existing terms we are not properly designed. Sometimes, it may not be possible to get the required information without the help of top management. If the information is vital to the system, we should make all possible help of top management.

- Accuracy

If the data is not accurate the output will be also not be correct.

CHAPTER-3

3. SYSTEM DESIGN

Search engine contains:

- Web Crawling
- Indexing
- Ranking

3.1 DATA FLOW DIAGRAM

A picture is worth a thousand words. A Data Flow Diagram (DFD) is traditional visual representation of the information flows within a system. A neat and clear DFD can depict a good amount of the system requirements graphically. It can be manual, automated, or combination of both.

It shows how information enters and leaves the system, what changes the information and where information is stored. The purpose of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communications tool between a systems analyst and any person who plays a part in the system that acts as the starting point for redesigning a system.

It is usually beginning with a context diagram as the level 0 of DFD diagram, a simple representation of the whole system. To elaborate further from that, we drill down to a level 1 diagram with lower level functions decomposed from the major functions of the system. This could continue to evolve to become a level 2 diagram when further analysis is required. Progression to level 3, 4 and so on is possible but anything beyond level 3 is not very common.

Now we'd like to briefly introduce to you a few diagram notations which you'll see in the tutorial below.

External Entity

An external entity can represent a human, system or subsystem. It is where certain data comes from or goes to. It is external to the system we study, in terms of the business process. For this reason, people used to draw external entities on the edge of a diagram.

Process

A process is a business activity or function where the manipulation and transformation of data takes place. A process can be decomposed to finer level of details, for representing how data is being processed within the process.

Data Store

A data store represents the storage of persistent data required and/or produced by the process. Here are some examples of data stores: membership forms, database table, etc.

Level 0:

These show the basic functionality information flow in the system. The webpage extracted from the URL server is submitted to the system which on referring databases like lexicon, URL's and forward index generates a reverse index which is referred by the searcher while providing results to the user.

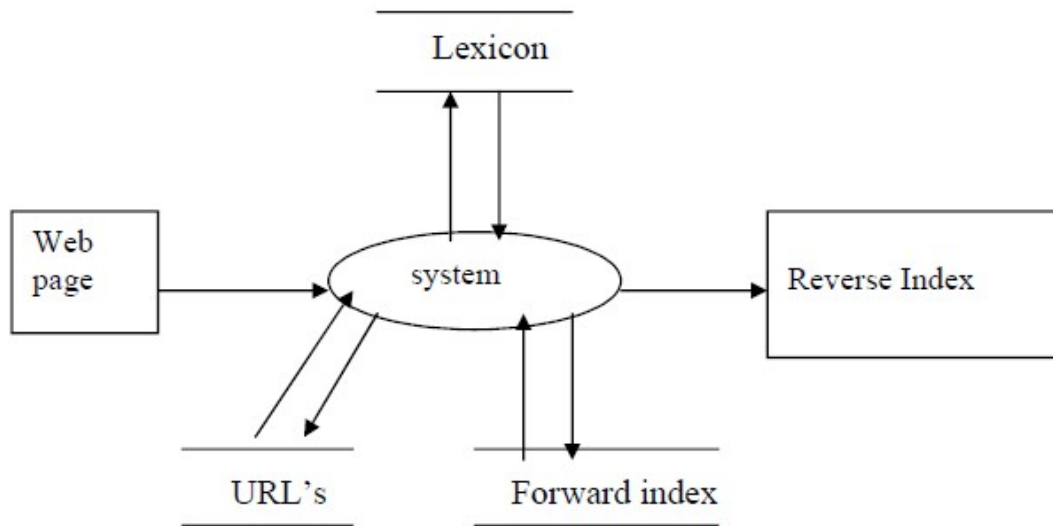


Figure 3.1

Level 1:

The process system is now broken down into various other processes in the LEVEL 2 DFD. The webpage is now given to the crawler which checks for the webpage accesses and if the access is given for a particular website then downloads the webpage and stores it in store server. The crawler also passes the webpage to the indexer. The indexer now refers to the lexicon for the words it scans in the webpage and then adds new words in the lexicon. It now prepares forward index. This forward index is now used by the sorter to give the reverse index.

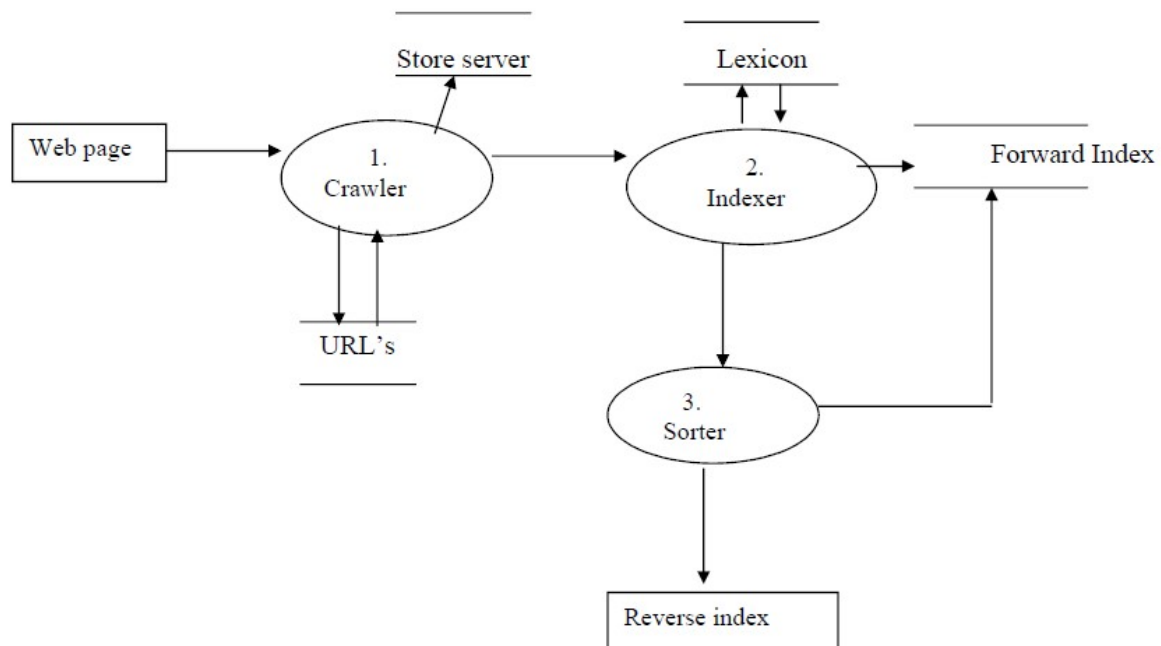


Figure 3.2

Level 2:

The processes are explained in more detail in LEVEL 2. The webpage is first scanned and URL database is referred. The animation sites in the given webpage is then listed and scanned for metatags. These sites are stored in store server. The tokens from the webpage are then obtained and added to lexicon if it is new token. The word id's are assigned to each tokens. The words are then stored in forward index along with various details like its location, font size and importance like underlined ,bold etc. The word id's are now considered one by one and docid's for each word id is retrieved from the forward index to prepare reverse index which stores word id and a sorted list of doc id's which contains the given word id.

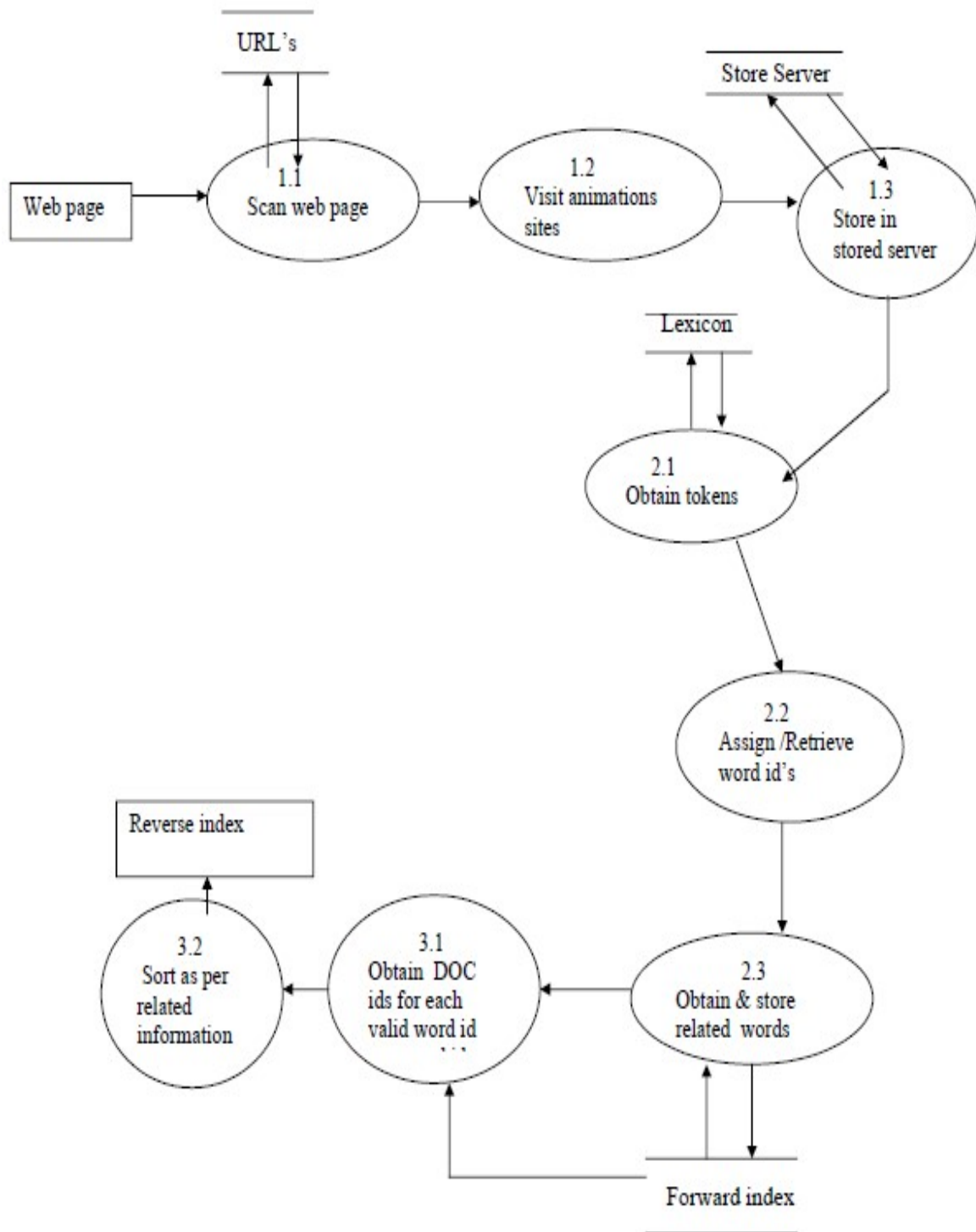


Figure 3.3

3.2 ENTITY RELATIONSHIP DIAGRAM (ERD)

This depicts relationship between data objects. The attribute of each data objects noted in the entity- relationship diagram can be described using a data object description. Data flow diagram serves two purposes:

1. To provide an indication of how data are transformed as they move through the system.
2. To depict the functions that transformation the data flow.

Data Objects: A data object is a representation of almost any composite information that must be understood by the software. By composite information, we mean something that has a number of different properties or attributes. A data object encapsulates data only there is no reference within a data object to operations that act on the data.

Attributes: Attributes define the properties of a data object and take on one of three different characteristics. They can be used to: Name an instance of data object. Describe the instance. Make reference to another instance in other table. 13 Relationships: Data objects are connected to one another in a variety of different ways. We can define a set of object relationship pairs that define the relevant relationships.

CARDINALITY AND MODALITY:

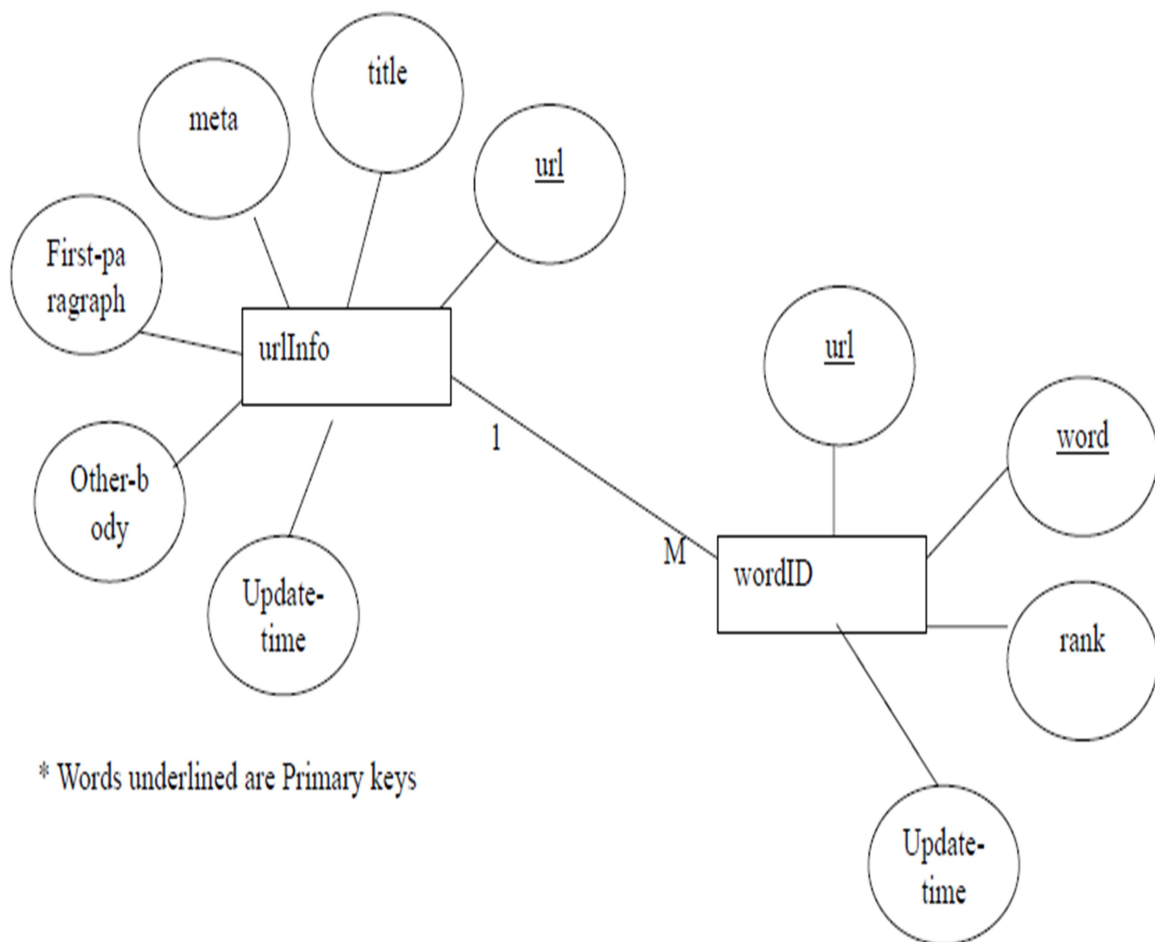
Cardinality: The data model must be capable of representing the number of occurrences of objects in a given relationship. The cardinality of an object relationship pair is

- ◆ One-To-One (1:1): An occurrence of object 'A' can relate to one and only one occurrence of object 'B' and vice versa.
- ◆ One-To-Many (1:N): One occurrence of object 'A' can relate to one or may occurrences of object 'B' but an occurrence of object 'B' can relate to only one occurrence of object 'A'.
- ◆ Many-To-Many (M: N): An occurrences of 'B' and an occurrence of 'B' can relate to one or many occurrence of 'A'.

Modality: The modality of a relationship is zero if there is no explicit need for the relationship to occur or the relationship is optional. The Modality is one if the occurrence of the relationship is mandatory. The object relationship pair can be represented graphically using the Entity Relationship Diagrams. A set of primary components are identified for the Entity Relationship Diagram,

1. Attributes,
2. Relationships
3. Various Type Indicators.

The primary purpose of the Entity Relationship Diagram is to represent data objects and their relationships.



Required Logical Data Model

Figure 3.4

ENTITY RELATIONSHIP DIAGRAM FOR SEARCH ENGINE

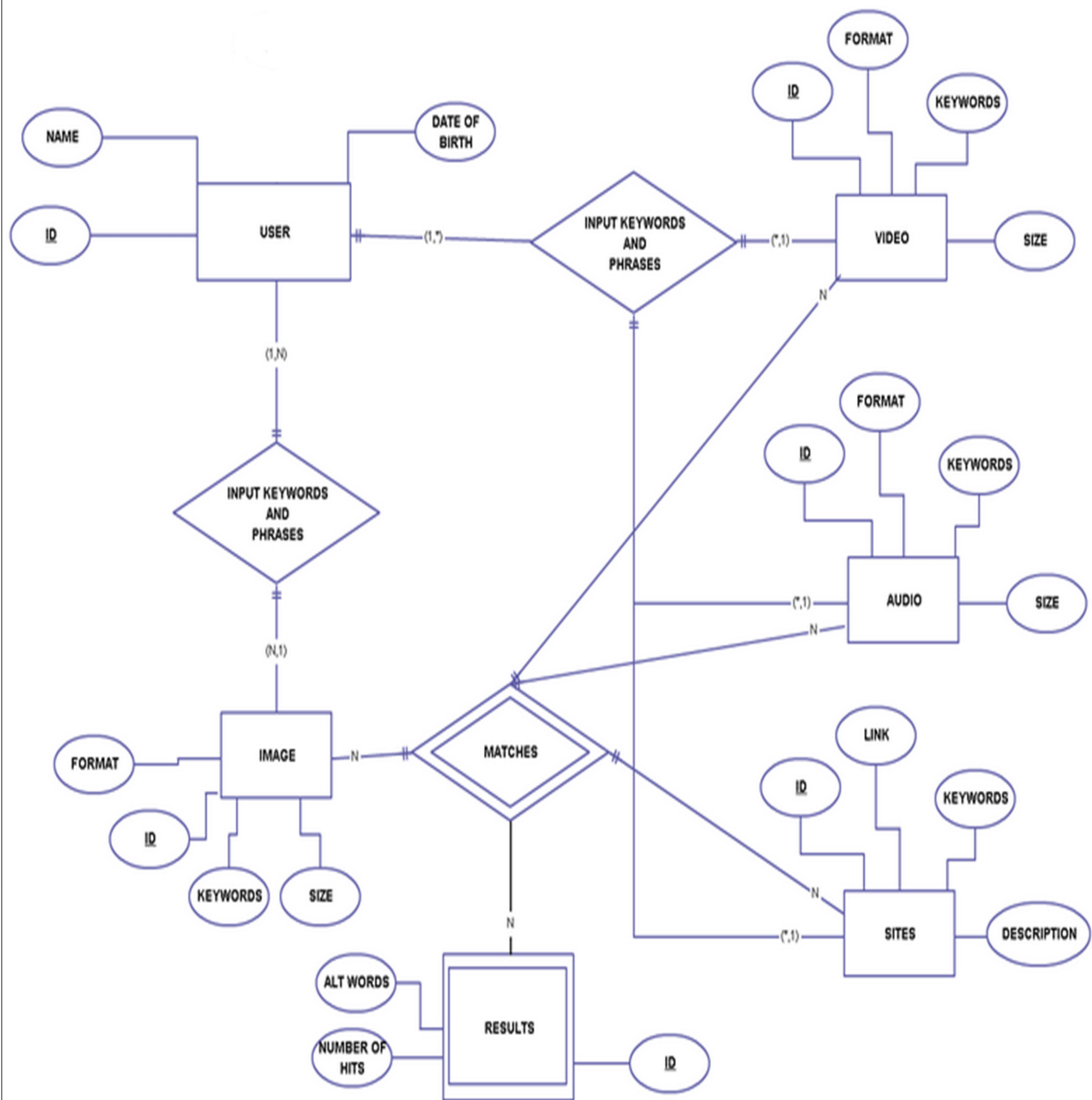


Figure 3.5

CHAPTER-4

4.CODING

```
import urllib
import socket
socket.setdefaulttimeout(10)
import logging
import robotparser
import urlparse
# in a true webcrawler you should not use re instead use a DOM parser
import re
# for sorting a nested list
from operator import itemgetter

def get_page(url):
    try:
        print "crawling:" + url
        f = urllib.urlopen(url)
        page = f.read()
        f.close()
        return page
    except:
        return ""
    return ""

def get_next_target(page):
    start_link = page.find('<a href=')
    if start_link == -1:
        return None, 0
    start_quote = page.find('"', start_link)
    end_quote = page.find('"', start_quote + 1)
    url = page[start_quote + 1:end_quote]
    return url, end_quote

def union(p,q):
    cnt = 0
    for e in q:
        if e not in p:
            p.append(e)
            cnt += 1
    return cnt

def get_all_links(page):
    #does not handle relative links
    links = []
    while True:
        url,endpos = get_next_target(page)
```

```

        if url:
            links.append(url)
            page = page[endpos:]
        else:
            break
    return links
def add_to_index(index,keyword,url):
    if keyword in index:
        if url not in index[keyword]:
            index[keyword].append(url)
    else:
        index[keyword] = [url]
def split_string(source,splitlist):
    return ".join([ w if w not in splitlist else ' ' for w in source]).split()
def add_page_to_index_re(index,url,content):
    i = 0
    # it is not a good idea to use regular expression to parse html
    # i did this just to give a quick and dirty result
    # to parse html pages in practice you should use a DOM parser
    regex = re.compile('(?!<script)[>](?![\\s\\#\\'<]).+?[<]')

    for words in regex.findall(content):
        word_list = split_string(words, """,!-.()<>[]{};:?!-='&""")
        for word in word_list:
            add_to_index(index,word,url)
            i += 1

    return i
def format_url(root,page):
    if page[0] == '/':
        return root + page
    return page
def crawl_web(seed,max_pages=10,max_depth=1):
    root = seed
    tocrawl = [seed]
    depth = [0]
    crawled = []
    index = {}
    graph = {}
    while tocrawl and len(crawled) < max_pages:
        page = tocrawl.pop()
        d = depth.pop()
        print "to crawl " + page
        if page not in crawled:
            #print "page not in crawled"
            page = format_url(root,page)
            content = get_page(page)

```

```

        success = add_page_to_index_re(index,page,content)
        outlinks = get_all_links(content)
        graph[page] = outlinks
        if d != max_depth:
            cnt = union(tocrawl,outlinks)
            for i in range(cnt):
                depth.append(d+1)
        crawled.append(page)
        print crawled
    return index, graph
def lookup(index, keyword):
    if keyword in index:
        return index[keyword]
    return None
def sort_by_score(l):
    get_score = itemgetter(0)
    map(get_score,l)
    l = sorted(l,key=get_score)
    l.reverse()
    return l
def lookup_best(index, keyword, ranks):
    result = []
    if keyword in index:
        for url in index[keyword]:
            if url in ranks:
                result.append([ranks[url], url])
    if len(result) > 0:
        result = sort_by_score(result)
    return result
def get_inlinks(page,graph):
    il = {}
    for p in graph:
        for ol in graph[p]:
            if ol == page:
                il[p] = graph[p]
    return il
def compute_ranks(graph):
    d = 0.8 # damping factor
    numloops = 10
    ranks = {}
    npages = len(graph)
    for page in graph:
        ranks[page] = 1.0 / npages
    for i in range(0, numloops):
        newranks = {}
        for page in graph:

```



```

        newrank = (1 - d) / npages
        inlinks = get_inlinks(page,graph)
        for il in inlinks:
            newrank += ((0.8 * ranks[il])/len(inlinks[il]))
        newranks[page] = newrank
    ranks = newranks
    return ranks
# code below only runs when this file is run as a script
# if you imported this code into your own module the code
# below would not be accessible by your code
if __name__ == "__main__":
    import os
    import pickle
    GLOBAL_NUM_SEARCHES = 8
    GLOBAL_TRENDING_INTERVAL = 4
    def calculate_trending(trending,now,before,interval,threshhold=0.5):
        for s in now:
            if s in before:
                slope = float(now[s] - before[s])/interval
                # set trending
                if slope > threshhold:
                    trending[s] = 1
                # clear trending
                if slope < 0:
                    if s in trending:
                        trending.pop(s)
            return trending
    def trending(searches,interval):
        curr_searches = {}
        prev_searches = {}
        is_trending = {}
        i = 0
        while(searches):
            search = searches.pop()
            if search in curr_searches:
                curr_searches[search] = curr_searches[search] + 1
            else:
                curr_searches[search] = 1
            i += 1
            if i == interval:
                is_trending
                calculate_trending(is_trending,curr_searches,prev_searches,interval)
                i = 0
                prev_searches = curr_searches.copy()
                curr_searches.clear()
        return is_trending

```

```

def print_cmds():
    return " Welcome to the singh Web Crawler\n" + \
        " What do you want to do?\n" + \
        " Enter 1 - To start crawling a web page\n" + \
        " Enter 2 - Print the Index\n" + \
        " Enter 3 - Find a word in the Index\n" + \
        " Enter 4 - Save Index\n" + \
        " Enter 5 - Load Index\n" + \
        " Enter 6 - Delete Index\n" + \
        " Enter q - Quit\n" + \
        " crawler:"

def execute_start_crawl(index):
    maxdepth = int(raw_input(" Enter Max Depth:"))
    maxpages = int(raw_input(" Enter Max Pages:"))
    url = raw_input(" Enter Web Url:")
    return crawl_web(url,maxpages,maxdepth)

def delete_file(path):
    ret = ""
    if os.path.exists(path):
        try:
            size2 = os.path.getsize(path)
            os.remove(path)
            ret += " Deleted {} ({} bytes)\n".format(path,size2)
        except:
            ret += " Failed to delete {}\n".format(path)
    print ret

def clear_data(data,data_str,path):
    ret = "
    delete_file(path)
    length = len(data)
    data.clear()
    ret += " Cleared {} entries from {}\n".format(length,data_str)
    print ret
    return data

def open_file(data,data_str,path):
    ret = ""
    file = open(path,"wb")
    if file:
        fail = 0
        try:
            pickle.dump(data,file)
        except:
            fail += 1
            ret += " Failed to save {} to {}\n".format(data_str,path)
    try:
        size = os.path.getsize(path)

```

```

except:
    size = 0
    fail += 1
    ret += "    Failed to get the size of {0}\n".format(path)

    if fail == 0:
        ret += "    {0} was saved to {0} ({0} bytes)\n".format(data_str,path,size)
        ret += "    {0} contains {0} entries.\n".format(data_str,len(data))
        file.close()
    else:
        ret += "    Failed to open {0} at {0}\n".format(data_str,path)
print ret
def load_file(data,data_str,path):
    ret = ""
    if os.path.exists(path):
        file1 = open(path,'rb')
        if file1:
            try:
                data = pickle.load(file1)
                size1 = os.path.getsize(path)
                ret += "    Loaded {0} from {0} ({0} bytes)\n".format(data_str,path,size1)
                ret += "    {0} contains {0} entries.\n".format(data_str,len(data))
            except:
                size1 = 0
                ret += "    Failed to load {0} from {0}\n".format(data_str,path)
        else:
            ret += "    Failed to open {0}\n".format(path)
    else:
        ret += "    {0} does not exist\n".format(path)
print ret
return data
def execute_cmd(c,index,graph, ranks, searches):
    if c == '1':
        index, graph = execute_start_crawl(index)
        ranks = compute_ranks(graph)
        print "    Crawl finished. Index has {0} items.".format(len(index))
        raw_input("    Press Enter")
        print ""
    elif c == '2':
        maxentries = raw_input("    Enter Number of Index Entries to Display (Type a for all):")
        if maxentries == 'a' or maxentries == 'A':

```

```

        maxentries = 0xFFFFFFFF
    else:
        maxentries = int(maxentries)
    for i, e in enumerate(index):
        if i >= maxentries:
            break
        print "    Entry {0}:".format(i)
        print "        '{0}' appears in the following urls:".format(e)
        for u in index[e]:
            print "            {0}".format(u)
    if len(index) == 0:
        print "    Index is empty"
    raw_input("    Press Enter")
    print ""
elif c == '4':
    open_file(index, "Index", os.getcwd() + os.path.sep + 'index.pkl')
    open_file(graph, "Graph", os.getcwd() + os.path.sep + 'graph.pkl')
    open_file(ranks, "Ranks", os.getcwd() + os.path.sep + 'ranks.pkl')
    raw_input("    Press Enter")
    print ""
elif c == '5':
    index = load_file(index, "Index", os.getcwd() + os.path.sep + 'index.pkl')
    graph = load_file(graph, "Graph", os.getcwd() + os.path.sep + 'graph.pkl')
    ranks = load_file(ranks, "Ranks", os.getcwd() + os.path.sep + 'ranks.pkl')
    raw_input("    Press Enter")
    print ""
elif c == '6':
    index = clear_data(index, "Index", os.getcwd() + os.path.sep + 'index.pkl')
    graph = clear_data(graph, "Graph", os.getcwd() + os.path.sep + 'graph.pkl')
    ranks = clear_data(ranks, "Ranks", os.getcwd() + os.path.sep + 'ranks.pkl')
    searches = []
    raw_input("    Press Enter")
    print ""
else:
    w = raw_input("    Enter a word to find from the index:")
    ret = ""
    if len(ranks) == 0:
        ranks = compute_ranks(graph)
    l = lookup_best(index, w, ranks)
    if len(l) == 0:
        ret = "        {0} was not found in index".format(w)
    else:
        ret += "        '{0}' appears in the following urls:\n".format(w)
        for e in l:
            ret += "            {0}\n".format(e[1], e[0])
            score =

```

```

        searches.append(w)
        is_trending = {}
        if len(searches) == GLOBAL_NUM_SEARCHES:
            searches.reverse()
            is_trending = trending(searches, GLOBAL_TRENDING_INTERVAL)
            searches = []
        if len(is_trending) > 0:
            ret += "    The following are trending:\n"
            for word in is_trending:
                ret += "        '{}'\n".format(word)
            print ret
            raw_input("    Press Enter")
            print ""
    return index, graph, ranks, searches
def main():
    index = {}
    graph = {}
    ranks = {}
    searches = []
    while(True):
        c = raw_input(print_cmds())
        if c == 'q' or c == 'Q':
            break
        index, graph, ranks, searches = execute_cmd(c, index, graph, ranks,
searches)
    main()

```

CHAPTER-5

5.TESTING

Testing is important because software bugs could be expensive or even dangerous. Software bugs can potentially cause monetary and human loss, history is full of such examples.

- In April 2015, Bloomberg terminal in London crashed due to software glitch affected more than 300,000 traders on financial markets. It forced the government to postpone a 3bn pound debt sale.
- Nissan cars have to recall over 1 million cars from the market due to software failure in the airbag sensory detectors. There has been reported two accident due to this software failure.
- Starbucks was forced to close about 60 percent of stores in the U.S and Canada due to software failure in its POS system. At one point store served coffee for free as they unable to process the transaction.
- Some of the Amazon's third party retailers saw their product price is reduced to 1p due to a software glitch. They were left with heavy losses.
- Vulnerability in Window 10. This bug enables users to escape from security sandboxes through a flaw in the win32k system.
- In 2015 fighter plane F-35 fell victim to a software bug, making it unable to detect targets correctly.
- China Airlines Airbus A300 crashed due to a software bug on April 26, 1994, killing 264 innocent live
- In 1985, Canada's Therac-25 radiation therapy machine malfunctioned due to software bug and delivered lethal radiation doses to patients, leaving 3 people dead and critically injuring 3 others.
- In April of 1999, a software bug caused the failure of a \$1.2 billion military satellite launch, the costliest accident in history
- In may of 1996, a software bug caused the bank accounts of 823 customers of a major U.S. bank to be credited with 920 million US dollars.

Types of Software Testing

Typically Testing is classified into three categories.

- Functional Testing
- Non-Functional Testing or Performance Testing
- Maintenance (Regression and Maintenance)

Functional Testing Techniques:

There are two major Functional Testing techniques as shown below:

- Black box testing
- White box testing

The other major Functional Testing techniques include:

- Unit Testing
- Integration Testing
- Smoke Testing
- User Acceptance Testing
- Localization Testing
- Interface Testing
- Usability Testing
- System Testing
- Regression Testing
- Globalization Testing

5.1 UNIT TESTING

Unit testing of software applications is done during the development (coding) of an application.

The objective of unit testing is to isolate a section of code and verify its correctness. In procedural programming a unit may be an individual function or procedure

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. Unit testing is usually performed by the developer.

Importance of Unit Testing

Sometimes software developers attempt to save time by doing minimal unit testing. This is a myth because skimping on unit testing leads to higher Defect fixing costs during System Testing, Integration Testing and even Beta Testing after the application is completed. Proper unit testing done during the development stage saves both time and money in the end.

Unit Testing Tools

There are several automated tools available to assist with unit testing. We will provide a few examples below:

- Rational Software - Rational Software by IBM has a unittest feature known as "Rational Test Realtime". The software contains a complete range of testing tools for much more than just unit testing. It is used for Ada, Java, C and C++. It creates unit tests by reverse engineering the software. Operating systems it supports include Windows, Linux, Solaris, HP-UX and AIX. Go to <http://www-01.ibm.com/software/rational/> to learn more.
- JavaScript Assertion Unit- Also known as jsAsserUnit, this Freeware JavaScript unit testing tool can be used on any platform that supports JavaScript. It is available at <http://jsassertunit.sourceforge.net/docs/index.html>
- CUT - CUT is a Freeware unittest tool for C, C++ and Objective C. It is great for embedded software testing frameworks and desktop applications on Linux and Windows operating systems. Learn more at sourceforge.net by going to <http://sourceforge.net/projects/cut/>.
- Dotunit - Dotunit is a .net framework Freeware unit testing tool. Part of Junit on the Microsoft .net framework, Dotunit is used for automating unit testing on windows systems. This is another tool from sourceforge.net, so look for it at: <http://dotunit.sourceforge.net/>.

Those are just a few of the available unit testing tools. There are lots more, especially for C languages and Java, but you are sure to find a unit testing tool for your programming needs regardless of the language you use.

5.2 INTEGRATION TESTING

We normally do Integration testing after "Unit testing".

Once all the individual units are created and tested, we start combining those "Unit Tested" modules and start doing the integrated testing. So the meaning of Integration testing is quite straightforward- Integrate/combine the unit tested module one by one and test the behaviour as a combined unit.

The main function or goal of Integration testing is to test the interfaces between the units/modules.

The individual modules are first tested in isolation. Once the modules are unit tested, they are integrated one by one, till all the modules are integrated, to check the combinational behaviour, and validate whether the requirements are implemented correctly or not.

Here we should understand that Integration testing does not happen at the end of the cycle, rather it is conducted simultaneously with the development. So in most of the times, all the modules are not actually available to test and here is what the challenge comes to test something which does not exist!

Need of Integration Testing

Although each software module is unit tested, defects still exist for various reasons like

- A Module in general is designed by an individual software developer whose understanding and programming logic may differ from other programmers. Integration testing becomes necessary to verify the software modules work in unity
- At the time of module development, there are wide chances of change in requirements by the clients. These new requirements may not be unit tested and hence system integration testing becomes necessary.
- Interfaces of the software modules with the database could be erroneous
- External Hardware interfaces, if any, could be erroneous
- Inadequate exception handling could cause issues.

Approaches:

- Big Bang is an approach to Integration Testing where all or most of the units are combined together and tested at one go. This approach is taken when the testing team receives the entire software in a bundle. So what is the difference between Big Bang Integration Testing and System Testing? Well, the former tests only the interactions between the units while the latter tests the entire system.
- Top Down is an approach to Integration Testing where top level units are tested first and lower level units are tested step by step after that. This approach is taken when top down development approach is followed. Test Stubs are needed to simulate lower level units which may not be available during the initial phases.

Stubs” can be referred to as code a snippet which accepts the inputs/requests from the top module and returns the results/ response. This way, in spite of the lower modules, do not exist, we are able to test the top module.

- Bottom Up is an approach to Integration Testing where bottom level units are tested first and upper level units step by step after that. This approach is taken when bottom up development approach is followed. Test Drivers are needed to simulate higher level units which may not be available during the initial phases.

In simple words, DRIVERS are the dummy programs which are used to call the functions of the lowest module in a case when the calling function does not exist. The bottom-up technique requires module driver to feed test case input to the interface of the module being tested.

The advantage of this approach is that, if a major fault exists at the lowest unit of the program, it is easier to detect it, and corrective measures can be taken.

The disadvantage is that the main program actually does not exist until the last module is integrated and tested. As a result, the higher level design flaws will be detected only at the end.

- Sandwich/Hybrid is an approach to Integration Testing which is a combination of Top Down and Bottom Up approaches.

5.3 SYSTEM TESTING

System_Testing is the testing of a complete and fully integrated software product.

Usually software is only one element of a larger computer based system. Ultimately, software is interfaced with other software/hardware systems. System testing is actually a series of different tests whose sole purpose is to exercise the full computer based system.

System testing involves testing the software code for following

- Testing the fully integrated applications including external peripherals in order to check how components interact with one another and with the system as a whole. This is also called End to End testing scenario..
- Verify thorough testing of every input in the application to check for desired outputs.
- Testing of the user's experience with the application. .

That is a very basic description of what is involved in system testing. You need to build detailed test cases and test suites that test each aspect of the application as seen from the outside without looking at the actual source code.

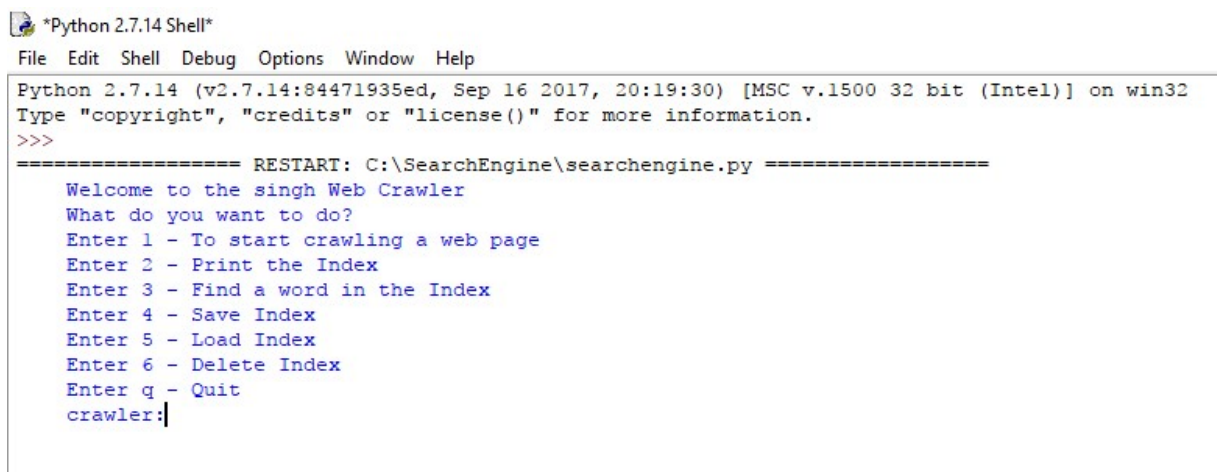
Different Types of System Testing

There are more than 50 types of System Testing. For an exhaustive list of software testing types click [here](#). Below we have listed types of system testing a large software development company would typically use

1. Usability Testing - Usability Testing mainly focuses on the user's ease to use the application, flexibility in handling controls and ability of the system to meet its objectives.
2. Load Testing - Load Testing is necessary to know that a software solution will perform under real-life loads.
3. Regression Testing- - Regression Testing involves testing done to make sure none of the changes made over the course of the development process have caused new bugs. It also makes sure no old bugs appear from the addition of new software modules over time.
4. Recovery Testing - Recovery testing is done to demonstrate a software solution is reliable, trustworthy and can successfully recoup from possible crashes.
5. Migration Testing - Migration testing is done to ensure that the software can be moved from older system infrastructures to current system infrastructures without any issues.
6. Functional Testing - Also known as functional completeness testing, Functional Testing involves trying to think of any possible missing functions. Testers might make a list of additional functionalities that a product could have to improve it during functional testing.
7. Hardware/Software Testing - IBM refers to Hardware/Software testing as "HW/SW Testing". This is when the tester focuses his/her attention on the interactions between the hardware and software during system testing.

Showing features of search engine:

It helps users to crawl in a web page and printing the index obtained from crawling and then allow them to find words in the index.



```
*Python 2.7.14 Shell*
File Edit Shell Debug Options Window Help
Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:19:30) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\SearchEngine\searchengine.py =====
Welcome to the singh Web Crawler
What do you want to do?
Enter 1 - To start crawling a web page
Enter 2 - Print the Index
Enter 3 - Find a word in the Index
Enter 4 - Save Index
Enter 5 - Load Index
Enter 6 - Delete Index
Enter q - Quit
crawler:|
```

Crawling:

User can enter the max depth and max no of pages upto which he want to carry out the searching.

```
Enter Max Depth:5
Enter Max Pages:4
Enter Web Url:https://www.amazon.com
to crawl https://www.amazon.com
crawling:https://www.amazon.com
['https://www.amazon.com']
to crawl /dogsofamazon
crawling:https://www.amazon.com/dogsofamazon
['https://www.amazon.com', 'https://www.amazon.com/dogsofamazon']
to crawl /gp/help/customer/display.html/ref=footer_cou?ie=UTF8&nodeId=508088
crawling:https://www.amazon.com/gp/help/customer/display.html/ref=footer_cou?ie=UTF8&nodeId=508088
['https://www.amazon.com', 'https://www.amazon.com/dogsofamazon', 'https://www.amazon.com/gp/help/customer/display.html']
to crawl }}}; }};
</script>

<script type='text/javascript'>
    window.$Nav && $Nav.declare('config.prefetchUrls', [
crawling:}}; }};
</script>

<script type='text/javascript'>
    window.$Nav && $Nav.declare('config.prefetchUrls', [
['https://www.amazon.com', 'https://www.amazon.com/dogsofamazon', 'https://www.amazon.com/gp/help/customer/display.html']
; }};\n</script>\n\n    <script type='text/javascript'>\n        window.$Nav && $Nav.declare('config.prefetchUrls', ["]
Crawl finished. Index has 2096 items.
Press Enter|
```

Printing the index entries:

```
Enter Number of Index Entries to Display (Type a for all):6
Entry 0:
    'Vitamins' appears in the following urls:
        https://www.amazon.com/gp/help/customer/display.html/ref=footer_cou?ie=UTF8&nodeId=508088
Entry 1:
    'Improvement' appears in the following urls:
        https://www.amazon.com/gp/help/customer/display.html/ref=footer_cou?ie=UTF8&nodeId=508088
Entry 2:
    'Dining' appears in the following urls:
        https://www.amazon.com/gp/help/customer/display.html/ref=footer_cou?ie=UTF8&nodeId=508088
Entry 3:
    '/movies' appears in the following urls:
        https://www.amazon.com/gp/help/customer/display.html/ref=footer_cou?ie=UTF8&nodeId=508088
Entry 4:
    '/climbing/b' appears in the following urls:
        https://www.amazon.com/gp/help/customer/display.html/ref=footer_cou?ie=UTF8&nodeId=508088
Entry 5:
    'Thrive' appears in the following urls:
        https://www.amazon.com/dogsofamazon
Press Enter|
```

Find a word in the index:

```
Welcome to the singh Web Crawler
What do you want to do?
Enter 1 - To start crawling a web page
Enter 2 - Print the Index
Enter 3 - Find a word in the Index
Enter 4 - Save Index
Enter 5 - Load Index
Enter 6 - Delete Index
Enter q - Quit
crawler:3
Enter a word to find from the index:Vitamins
    'Vitamins' appears in the following urls:
        https://www.amazon.com/gp/help/customer/display.html/ref=footer_cou?ie=UTF8&nodeId=508088
        score = https://www.amazon.com/gp/help/customer/display.html/ref=footer_cou?ie=UTF8&nodeId=508088

Press Enter|
```

6.CONCLUSION

This project is designed to meet the requirements of searching all the hyperlinks in a seed page. It has been developed by using python, keeping in mind the specifications of the system. For designing the system we have used simple web crawling, indexing and page ranking techniques. Overall the project teaches us the essential skills like: Using crawler and ranking techniques in designing the system.

The system will be able to search for hyperlinks in a page and will show indexes. It will show all the hyperlinks with the given criteria, the system will allow to search for indexes and you can select the depth and upto how many pages you want to search.

CHAPTER-7

7.FUTURE SCOPE

Imagine an operating system that monitors all of your activities – with your permission, of course. Every file, every image, word document, mp3, even e-books could be monitored by your computer as it endeavors to anticipate your every need. Not only could an integrated search engine allow you to search files located on your hard drive, but it could also use the information it has collected from these files to make your online search experience even more enjoyable. When you think about the future of search, it is easy to get excited. Millions (if not billions) of dollars are going to be filling the coffers of the largest search engine providers. They have some of the smartest people in the world working to develop the next great “thing”, which will enhance the user experience and serve up better, more relevant search results. Search engine technology is still most definitely in its infancy; how it grows will very much depend upon how much information and privacy the average search engine user is willing to give up. Personally, if I can view search results that more closely match my desired results

CHAPTER-8

8.REFERENCES

- <https://www.udacity.com>
- <http://www.serachtools.com>
- <http://www.press.umich.edu/jep/07-01/bergman.com>
- <http://www.robotstxt.org>
- <http://www.archives/eprints.org>
- http://en.wikipedia.org/wiki/Web_crawler
- <http://www.searchengineshowdown.com/features/google/review.html>
- http://en.wikipedia.org/wiki/Search_engine
- <http://www.openarchives.org/registar/browsesites>