AJCSE
ASIAN JOURNAL OF COMPUTER SCIENCE ENGINEERING

www.ajcse.info

Asian Journal of Computer Science Engineering 2017; 2(6):08-11

REVIEW ARTICLE

Reviewing Schematic Model of Search Engine

***Muhammad Fuzail[1], Anum Aftab[2]**

**[1,2]Department of Computer Science and Engineering, University of Engineering and Technology, Lahore, Pakistan.*

Received on: 28/04/2017, Revised on: 08/07/2017, Accepted on: 28/07/2017

ABSTRACT

In this paper we discuss the user interfaces of search engine by using different language like HTML etc & also we discuss design goals of search engine. We also discuss the lorel language for back-end design of search engine and for the search engine basic searching we use different algorithm. Here we discuss soundex algorithm for searching.

INTRODUCTION

The web creates new challenges for information retrieval. The amount of information on the web is growing rapidly, as well as the number of new users inexperienced in the art of web research. People are likely to surf the web using its link graph, often starting with high quality human maintained indices such as Yahoo! or with search engines. Human maintained lists cover popular topics effectively but are subjective, expensive to build and maintain, slow to improve, and cannot cover all esoteric topics.

Automated search engines that rely on keyword matching usually return too many low quality matches. We have built large databases to store the information of search engines. How these databases are created we see it by using Lore Language. This paper describes the implementation of Lore language designed specifically for managing semi-structured data. We use different search algorithms to do the searching as search engines play a vital role in effective searching and browsing querying.

WEB SEARCH ENGINE

Search engine technology has had to scale dramatically to keep up with the growth of the web. In 1994, one of the first web search engines, the World Wide Web Worm (WWWW) [McBryan 94] had an index of 110,000 web pages and web accessible documents. As of November, 1997, the top search engines claim to index from 2 million (WebCrawler) to 100 million web documents (from Search Engine Watch). It is foreseeable that by the year 2000, a comprehensive index of the Web will contain over a billion documents. At the same time, the number of queries search engines handle has grown incredibly too. In March and April 1994, the World Wide Web Worm received an average of about 1500 queries per day. In November 1997, AltaVista claimed it handled roughly 20 million queries per day. With the increasing number of users on the web, and automated systems which query search engines, it is likely that top search engines will handle hundreds of millions of queries per day by the year 2000. The goal of our system is to address many of the problems, both in quality and scalability, introduced by scaling search engine technology to such extraordinary numbers.

DESIGN GOALS

IMPROVED SEARCH QUALITY

Our main goal is to improve the quality of web search engines. In 1994, some people believed that a complete search index would make it possible to find anything easily. According to Best of the Web 1994 -- Navigators, "The best navigation service should make it easy to find almost anything on the Web (once all the data is entered)." However, the Web of 1997 is quite different. Anyone who has used a search engine recently can readily testify that the completeness of the index is not the only factor in the quality of search results. "Junk results" often wash out any results that a user is interested in. In fact, as of November 1997, only one of the top four commercial search engines finds itself (returns its own search page in response to its name in the top ten results). One of the main causes of this problem is that the number of documents in the indices has been increasing by many orders of magnitude, but the user's ability to look at

***Corresponding Author:** Muhammad Fuzail**, Email**: m.fuzail@ymail.com

documents has not. People are still only willing to look at the first few tens of results.

Because of this, as the collection size grows, we need tools that have very high precision (number of relevant documents returned, say in the top ten of results). Indeed, we want our notion of "relevant" to only include the very best documents since there may be tens of thousands of slightly relevant documents. This very high precision is important even at the expense of recall (the total number of relevant documents the system is able to return). There is quite a bit of recent optimism that the use of more hyper textual information can help improve search and other applications [Marchiori 97] [Spertus 97] [Weiss 96] [Kleinberg 98]. In particular, link structure [Page 98] and link text provide a lot of information for making relevance judgments and quality filtering. Google makes use of both link structure and anchor text.

ACADEMIC SEARCH ENGINE RESEARCH

Aside from tremendous growth, the Web has also become increasingly commercial over time. In 1993, 1.5% of web servers were on .com domains. This number grew to over 60% in 1997. At the same time, search engines have migrated from the academic domain to the commercial. Up until now most search engine development has gone on at companies with little publication of technical details. This causes search engine technology to remain largely a black art and to be advertising oriented. With Google, we have a strong goal to push more development and understanding into the academic realm.

Another important design goal was to build systems that reasonable numbers of people can actually use. Usage was important to us because we think some of the most interesting research will involve leveraging the vast amount of usage data that is available from modern web systems. For example, there are many tens of millions of searches performed every day. However, it is very difficult to get this data, mainly because it is considered commercially valuable. Our final design goal was to build an architecture that can support novel research activities on large-scale web data. To support novel research uses, Google stores all of the actual documents it crawls in compressed form. One of our main goals in designing Google was to set up an environment where other researchers can come in quickly, process large chunks of the web, and produce interesting results that would have been very difficult to produce otherwise. In the short time the system has been up, there have already been several papers using databases generated by Google, and many others are underway.

Another goal we have is to set up a Spacelab-like environment where researchers or even students can propose and do interesting experiments on our large-scale web data.

This is all about the design goals of search engine. Now we discuss the back-end design of search engine.

THE LOREL QUERY LANGUAGE

Lore is specifically for handling semi-structured data. In this we use the query same as SQL but in this one should not worry about the irregularities of data type. It uses OEM (Object Exchange Method). In OEM objects are written in the form of directed graphs where objects are vertices with Object Identifiers (OID). In this information is not fixed and can be changed dynamically. In OEM, database is self-describing that is there is no regularity imposed on data. Lorel offers a richer form of "declarative navigation" .In OEM databases than simple path expressions, namely general path expressions. It means user can specify desired pattern of labels in the database: one can specify patterns for paths (to match sequences of labels), pattern for labels (to match sequences of characters), and pattern for atomic values.

SYSTEM ARCHITECTURE OF LOREL LANGUAGE

The basic architecture of the Lore system is depicted in Figure. This section gives a brief introduction to the components that make up Lore. More detailed discussions of individual components appear in subsequent sections.

Access to the Lore system is through a variety of applications or directly via the Lore Application Program Interface (API). There is a simple textual interface, primarily used by the system developers, but suitable for learning system functionality and exploring small databases.

The graphical interface, the primary interface for end-users, provides powerful tools for browsing query results, a Data Guide feature for seeing the structure of the data and formulating simple queries \by example," a way of saving frequently asked queries, and mechanisms for viewing the multimedia atomic types such as video, audio, and java. These two interface modules, along with other applications, communicate with Lore through the API.

The Query Compilation layer of the Lore system consists of the parser, preprocessor, query plan

generator, and query optimizer. The parser accepts a textual representation of a query, transforms it in to a parse tree, and then passes the parse tree to the preprocessor. The preprocessor handles the transformation of the Lorel query into an OQL-like query. A query plan is generated from the transformed query and then passed to the query optimizer. In addition to doing some (currently simple) transformations on the query plan, the optimizer also decides whether the use of indexes is feasible. The optimized query plan is then sent to the Data Engine layer.

The Data Engine layer houses the OEM object manager, query operators, external data manager, and various utilities. The query operators execute the generated query plans. The object manager functions as the translation layer between OEM and the low- level constructs. It supports basic primitives such as fetching an object, comparing two objects, performing simple coercion, and iterating over the sub-objects of a complex object. In addition, some performance features, such as a cache of frequently accessed objects, are implemented in this component. The index manager, external data manager, and Data Guide manager. Finally, bulk loading and physical object layout on disk.

This is all about the back-end functionality of the search engine now we will discuss the search algorithms in detail.

ALGORITHM

An algorithm is a procedure to solve a problem in an orderly, finite, step-by-step logical and
Straight forward manner. An algorithm must be finite (terminating after a finite number of steps) and be effective (accomplish the solution of the problem in a series of simple steps).

Here some of the important search algorithms are discussed.

- Soundex algorithm
- Metaphone algorithm
- Phonex algorithm
- Word Stemming Algorithms

SOUNDEX ALGORITHM

Margaret O' Dell and Robert C. Russell patented the original soundex algorithm in 1918 (1). The method is based on the six phonetic classifications of human speech sounds, which in turn are based on how you put your lips and tongue to make the sounds.

1. **Bilabial:** Sound produce with both lips.
2. **Labiodentals:** Utter with the participation of the lip and the teeth (the labiodentalssounds \f\ and \v\).
3. **Dental:** Articulated with the tip or blade of the tongue against or near the upper front teeth.
4. **Alveolar:** Articulated with the tip of the tongue touching or near the teeth ride.
5. **Velar:** Formed with the back of the tongue touching or near the soft palate (The velar \k\of \kill\ cool).
6. **Glottal:** the interruption of the breath stream during speech by closure of the glottis.

Terms that are often misspelled can be a problem for database designers. Names, for example, are variable length, can have strange spellings, and they are not unique. Words can be misspelled or have multiple spellings, especially across different cultures or national sources. To solve this problem, we need phonetic algorithms, which can find similar sounding terms and names. Such families of algorithms exist and are called Soundex, after the first patented version.

The Soundex algorithm is used for search of words on the basis of their phonetic quality. So the results will contain words that might have different spellings and yet sound similar when spoken.

This algorithm is useful to search for keywords when the exact spelling is unknown. A Soundex algorithm takes a word, such as a person's name, as input and produces a character string, which identifies a set of words that are (roughly) phonetically alike. It is very handy for searching large databases when the user has incomplete data. The algorithm is fairly straightforward to code and requires no backtracking or multiple passes over the input word. Following is the outline of soundex algorithm.

1. Capitalize all letters in the word and drop all punctuation marks. Pad the word with rightmost blanks as needed during each procedure step.
2. Retain the first letter of the word.
3. Change all occurrences of the following letters to '0' (zero):'A', E', 'I', 'O', 'U', 'H', 'W', 'Y'. (Except when they come as an initial letter)
4. Change letters from the following sets into the digit given:
    a. 1 = 'B', 'F', 'P', 'V'
    b. 2 = 'C', 'G', 'J', 'K', 'Q', 'S', 'X', 'Z'

c. 3 = 'D','T'
d. 4 = 'L'
e. 5 = 'M','N'
f. 6 = 'R'

5. Remove all pairs of digits, which occur beside each other from the string that resulted after step 4.
6. Remove all zeros from the string that results from step 5.0 (placed there in step 3)
7. Pad the string that resulted from step (5) with trailing zeros and return only the first fourpositions, which will be of the form <uppercase letter><digit><digit><digit>.

Example

Steps 0 1 2 3 4 5 6

William's à WILLIAMS à W à W0LL00MS à W0440052 à W04052 à W4527à**W452** would be the soundex code for word WILLIAM'S.

REFERENCES

8. Lakshmi, K. V. (2002). Developing a word-stemming program using Porter's Algorithm,from http://144.16.72.189/opendl/cdrom/test-collections/gsdl/project-reports/Lakminor/lakproj.ppt.
9. The Porter Stemming Algorithm.from http://www.tartarus.org/~martin/PorterStemmer/voc.txt
10. An algorithm for suffix stripping. Fromhttp://www.tartarus.org/~martin/PorterStemmer/def.txt
11. The Soundex Indexing System. From http://www.archives.gov/research_room/genealogy/census/soundex.html
12. Merriam-Webster Unabridged Expands. from http://www.m-w.com/home.htm
1. UnderstandingClassicSoundExAlgorithms fromhttp://www.creativyst.com/Doc/Articles/SoundEx1/SoundEx1.htm/
2. Franki, P., & Leonard, S. (2001). Is soundex good enough for you? From http://www.onomastix.com/about_ono/whitepapers/wp_soundex.htm/
3. Lang, R. A. (2001). The Compact Disc Search Engine, from http://www.rlang.co.uk/projects/soundz/Soundz.html/
4. WWW Search Engine Software. from http://www.Htdig.org/
5. Greenstone digital library software. from http://www.greenstone.org/
6. Hardy, D. R., & Lee, K.-J. (2002). Harvest User's Manual, fromhttp://www.harvest.sourceforge.net/harvest/doc/html/manual.html/
7. What is Stemming?from http://www.comp.lancs.ac.uk/computing/research/stemming/general/

