CHAPTER 3
MULTILEVEL HASHING FOR FAST ACCESS

3.1 INTRODUCTION

Fast IR is an imperative task in data management, substantially due to the increasing availability of data over the internet. In the past few decades, the process of information retrieval and the database communities have been a great confrontation due to the proliferating availability of complex data from different domains. Hence, representing and searching complex data over Internet has become a non-trivial task in the domain of web mining. In order to make the information retrieval process more facile over the Internet, the work based on multilevel hashing with indexing is discussed in this section, which enables fast access to documents.

Moreover, electronic resource is a collection of data such as information representing text, numbers, images, videos and graphics. The diversity and enormous amount of information makes search and retrieval of relevant information from web documents a challenging task. Further, it is to be stated that the user can access a web page in any one of the following ways:

- Directly accessing the location of the web page
- Hyperlinks on web page as related information
- Based on the user profile “narrow cast” service push web pages
- Finally, through search engines
The search engines collect data in two ways:

1. Companies or individuals can create and update their web page directly
2. The web robots or web spiders are traversed among web pages to search for new web documents or data

As a result, there exist vast amount of information available from which retrieving relevant information quickly with accuracy is much more difficult. Hence, there is a need for maintaining the content of web documents in store aware format in a dynamic way. In addition, the document content must be stored in a way for fast access to the relevant query. When a user submits a query to the search engine, it searches its documents in its database and returns a list of web pages, which are arranged by some relevant criteria. Based on these criteria, search engines try to rank the web pages in association with the query word. Figure 3.1 depicts the typical Information Retrieval System on Web Mining Process.

Figure 3.1 A typical IR system on web mining
Search engines follow special mechanism named information retrieval, which is the task to represent, store, archive and retrieve the relevant web documents from the Internet. Many research works are under progress to improve the access performance. The trend is going towards semantic web search engine though it has a lot of overhead of storing relevant information from the web for the documents. However, to aid an efficient algorithm for fast accessing of the web documents, one needs a pre-processing step. Therefore in this chapter a data structure, based on two-stage multilevel hashing is proposed. This assists in indexing process. The system builds a database named BuildKeyDB which is used for query searching. It uses two-stage multilevel hashing method to index the documents depending on the frequency of the keyword. Computational analysis shows that the proposed new data structure is efficient in retrieving documents.

In order to acquire appropriate and more relevant results, the query processing has to be done effectively. The queries given by the user may comprise single words, phrases, questions, whole passages and documents. For delivering superior results in a consistent manner, the exact intent of the query should be clearly defined. Moreover, the analysis of search query has been effectively made with the process named Query and Result (QR) processing. Basically, QR processing is the application of algorithm over the original query or to the raw results forwarded by the search engine. Generally, the performance of query processing is determined in terms of processed Query per Second (QPS) and by the grasped relevancy of the results. The usage of query processing completely depends on the complexity of the content, business model and search goals. While analyzing the information retrieval system, speed is the significant parameter to be considered. If the retrieval system is very slow in extracting the relevant information, it will be expendable, regardless of its ability to spot the relevant documents. Moreover, the characteristics of the typical users are that they do not want to wait for the
responses, and the recent drifts in the volume and the availability of data imply that speed will become more significant.

Classical IR uses keywords that pave the way for fast IR. The major limitation of this method lies in the user acquiring the information he/she needs, as keywords are not adequate for effectively directing a search process. In order to overcome those limitations, the proposed work incorporates the multilevel hashing for indexing the web documents in a prioritized manner.

For retrieving related information from internet, the work initiates the process of acquiring the queries through the search engines effectively. The concern may be a single word or a phrase that will be banded with the results made by initial operations of IR. In this proposal, terms are unbreakable alphanumeric strings entered by the user. They include words abbreviations, numbers and logical operators (AND, OR, NOT). E-mails and URLs are treated as single terms. It is also to be claimed that the link and URL information of the Internet are much needed to compensate the insufficiencies of the content information. Hence, the user will be satisfied with acquiring data with high goodness value. The search queries are the integral component for both the online database searching and web searching.

Generally, the queries are classified under two categories, namely topic relevance task-based query and homepage finding task-based query, depending on the question type of the query. Further in this work, two types of queries are considered.

1. Unique queries
2. Repeat queries
Unique queries are all different queries requested by a single user in a single session. Repeat queries are all multiple occurrences of the same query terms in one session. A session is the entire set of queries given by the same user over time. After getting the queries, the significant words (mentioned as keywords (KWD)) are extracted, which are given for indexing. Appropriate similarity search algorithm is needed for acquiring the results with high precision. Hence, the evaluation of queries can be effectively done with the employment of indexing structures. The multilevel hashing is accomplished with indexing the keywords data in web documents.

Accessing the data using keywords is a central operation in information retrieval from the Internet. An information retrieval process must evaluate, given a particular query and the resource location of the information that is relevant to the document. Typically, hashing is a technique that provides ubiquitous information retrieval strategy for affording efficient information access based on the keywords. Based on some concerns, hashing is valuable both in time and space. It describes that the information can be acquired in constant time, whereas the space is less acceptable in most circumstances.

Moreover, hashing effectively implements the mapping process for domain of keywords to the domain of locations as per the user concern, in considerable speed. Hash tables are maintained to store the information retrieved, whereas a hash table is the thought of an array of locations called buckets. Each bucket is indexed by the integer values. If a bucket is empty, it states that there is no key in use, not empty means, corresponds to a key in use. Hash function is a function that is liable for mapping between a key and a bucket. Furthermore, hash function is a function whose domain is that of the keys and whose range lies between the assigned integer values. It is also responsible for storing the values in the hash table for computing the bucket
to which a key corresponds and placing the information in that bucket. The retrieving information needs hash functions to determine the bucket that comprises it. Evaluation of query under each index organization is quite different. Here, the processing has been made with the multilevel hashing based technique. In this chapter, the basis of multilevel hashing is discussed in Section 3.2, followed by the fast access to the web documents in Section 3.3. Section 3.4 analyses the computational complexity of the proposed work with the results discussed in Section 3.5.

### 3.2 GENERAL DESCRIPTION OF MULTILEVEL HASHING

In general, multilevel hashing is divided into two parts.

1. Directory
2. Bucket

Buckets are used to store the full hash keys of and the pointers to the indexed data items. The bucket has been determined into which a data item is appended based on the prefix $h_s$ of $s$ bits of the hash key $h$. Each possible bit combination of the prefix can find an entry in the directory, pointing to the corresponding bucket. At an instance, the directory has $2^s$ entries, where $s$ is called the global depth. Due to some excessive entries, a bucket may overflow. In such cases, the buckets get split into two and all their contents are divided among the two resulting buckets.

For determining the new storage space of a data item, the length of the inspected hash key prefix has to be enlarged till two data items acquire different hash key prefixes. The size of the current increased hash key prefix is denoted as $s'$ (of a bucket), which is called local depth. After a split, if the local depth $s'$ of a bucket is larger than the global depth $s$, the size of the directory has to be increased. This can be accomplished by simply doubling
the directory based on the requirement to have a new global depth $s$ should be equal to the local depth $s'$. The hash tables share pages by using the multilevel extensible hash tree in accordance with the buddy scheme. In this scheme, hash tables are represented as $y$-buddies that reside on the same page, in which the stored hash key shares a prefix of $y$ bits. Specifically, all buddy hash tables in the hash tree have the same global depth $y$.

For example, the generic illustration for indexing is given here. Let us assume that a page can hold $2^n$ entries of a hash table directory. Moreover, it is also assumed that the top-level hash table directory has been filled already that contains $2^n$ different entries at an 'n' instance. With that concern, a new hash table of global depth 1 is allocated to differentiate the element in the former bucket with respect to their $n+1^{th}$ bit. However, this can be done not only for overflowing bucket, but also for all buddies of this bucket. The hash tables for the remaining buddies are predicted in anticipation of further splits. Figure 3.2 denotes the structure in which the hash tables are allocated on a single page.

![Figure 3.2 Overflow in multilevel hash tree](image)

**Figure 3.2 Overflow in multilevel hash tree**
If another overflow occurs in anyone of the stored hash tables on level 2, considerably the global depth of all hash tables are increased on this page by 1 and doubling their directory size. At this moment, two pages are needed to store these tables, so the original page is split into two and the content that does not fit to a new page is copied. The next task is adjusting the pointers in the parent directory. The demonstration is given in Figure 3.3, the left half of the pointers directing the original page and the right half to the new page.

Further, the space utilization of the index can be effectively enhanced by eliminating the pages having unnecessary hash tables. When there is a new data item to be inserted on the hash table, the bucket has to be found for storing the new data item. The work of second level hashing revealed in Figure 3.3 has been carried out for all the levels for optimally indexing the data items. The generic scenario is enforced with the keyword-based manner in the proposed system.

Figure 3.3 Overflow on the second level of hashing
Moreover, multilevel hashing provides appropriate results in searching web documents in a fast manner than the non-hierarchical process of hashing. The difference between the non-hierarchical hashing scheme and the multilevel approach has been demonstrated through an example. Consider $S$ as a non-hierarchical hash table with a global depth value as 4. The requirement here is to produce the superset suffixes of the given query set $Q$ for $S$; eight superset prefixes are produced, namely 0010, 0011, 0110, 1010, 0111, 1011, 1110 and 1111. Hence, for a non-hierarchical hash table, eight sub queries are started here to access three of the seven buckets as in Figure 3.4.

Figure 3.4 Non-hierarchical hash table

For multilevel hashing approach, the top level superset subsets are assigned as 00, 01, 10 and 11 and then, as revealed in Figure 3.5, the superset 1 for the hash table on the left hand side of the second level of hashing, followed by the superset 10 and 11 for the hash table on the right hand side of the second level. Hence, seven superset subsets are needed to generate instead of eight at the first view.
Moreover, it is to be claimed from the above example that the multilevel hashing index can handle the stored data much better than the other dynamic hashing schemes resulting in a much smaller directory. The next section describes the fast access of web documents using multilevel hashing.

3.3 FAST ACCESS OF WEB DOCUMENTS USING MULTILEVEL HASHING

As there are massive amount of information available on the Internet, there arises a difficulty in terms of finding the location of the required information. Hence, the search engine requires proper maintenance of web documents. In that concern, as mentioned earlier, this work focuses on enforcing efficient multilevel hashing mechanism on indexing for the fast access of required information from the Internet. Further, the work has been framed as two stages.

3.3.1 First Stage

The first stage of phase1 work has been focused on fast retrieval of information by developing a database. The database, created for fast retrieval
of information is named as BuildKeyDB. Moreover, it involves performing two-level of multilevel hashing. The most commonly used file structures for information retrieval are classified into three categories.

- Lexicographical indices
- Clustered file structures
- Indices based on hashing

In the category of indices based hashing, each document is assigned a list of keywords with optional weights associated with that. Then, each keyword is directed with a link to the required document, which is relevant to that keyword. Here, the indexing has been made with the multilevel hashing.

The multilevel hashing data structure is shown in Figure 3.6 which is used for building BuildKeyDB database effectively. Here, the hashing process is analyzed with two levels. The first level of hashing resides on the indexing of keywords, whereas the second level of hashing is made with the URL of the related documents.

![Two level hashing structure for storing URL table with frequency of keywords](image)

**Figure 3.6** Two level hashing structure for storing URL table with frequency of keywords
Level 1

Indexing of keywords is carried out in level one of multilevel hashing. Keywords are read from web documents. Each keyword is associated with the collection of records, specific to particular domain knowledge. Each record corresponds to one URL of the web document and the frequency of the keywords in this URL. The frequency of the keyword $KWD_i$ is given by

$$freq_j(KWD_i) = \sum_{j=1}^{\alpha} p_{i,j}$$ (3.1)

Equation (3.1) denotes that the keyword $KWD_i$ has the probability of $p_{i,j}$ to appear in the $j^{th}$ document. Moreover, the sorted array structure is used to store the list of keywords in an organized manner. It also comprises the number of documents associated with each listed keyword and the link of the document that contains the keyword.

Level 2

The keyword frequency along with the URL is stored as the second level hash structure. The frequency has obtained the significance of the keyword term within the document. For building this collection, each web document is read one after another. Each keyword in a document is filtered through the stop words. For every access to the second level hash structure for a given keyword, either it enters the URL entry with frequency count of one or it increments the frequency count by one depending on whether the URL containing this keyword is already existing in the collection. This building of the keyword database is done offline with respect to the query searching. It is equivalent to the result of web crawling and update key word database.
3.3.2 Second Stage

Once the keyword database is built, it is ready for deployment for query search. However, the second level hash is indexed by URLs; it is not applicable for the keyword-based ranking, as URL indexes it. Hence, the second level hashing cannot be used directly for fast retrieval. Thus, another hashing table indexed by keyword is constructed and it in Figure 3.7. The associated value for the keyword is the organized table of URL and frequency has also been sorted. The sorted array has been produced on the basis of stored documents. Further, the process is accomplished in three steps.

Step 1: Parsing the input text into a list of words along with their location in the text

Step 2: Sorting the keyword list

Step 3: Reorganizing or compressing the files (optional)

Figure 3.7 Hashing structure with elements as URL table sorted by frequency
For creating the initial keyword list, different operations are needed. First, the individual words should be reorganized from the text. Then, each word is audited against the stop word list of common terms. If it is found that the particular word is a non-common word, it will be passed over for the stemming process to formulate the efficient keywords for effective searching mechanism. In the proposed work, the sorting process is accomplished based on the keywords. It is obvious that the sorting is a time consuming process for large data sets. An efficient way to handle this problem is breaking the large data sets into smaller pieces, processing each separately and then merging the results.

Thus, the proposal has two stages of operations. The first stage builds keyword database. During the second stage, the query word is indexed in the hash, and the sorted table is accessed for ranking the web pages. The Algorithm ‘BuildKeyDB’ outlines the process of the building the two-level hash data structure as per the Figure 3.6 and Figure 3.7.

Algorithm for BuildKeyDB

```
Algorithm: BuildKeyDB()
//Input: Web document corpus
//Output: Two level hashing structure with URL //Table
{
    for each URL u
        for each content keyword Ki
            If ( !isMemberStopwords(Ki) )
                Store (H, Ki, u, updateFreq(freq));
            H = sortByFreq(H);
}
```
During the query process, the given keyword is searched in the hash, and the web documents are ranked and delivered to the user with minimal retrieval time, since the hash elements are already sorted by frequency. The search time complexity is decided only by the hashing function, which is not an iterative process.

As stated above, web proposes a great confrontation for locating quality information. Moreover, the ease of content distribution on the web tends to enormous diversity of the accessible data regarding topic, quality, presentation etc., along with the emerging growth of available data. The web documents that are relevant to the user query are to be viewed separately. The quality and the relevancy rate of the retrieved web document are to be considerably high and are to be ranked in that manner. Hence, it is important to improve the ability of users to locate the relevant information with high quality web documents. The next step leads to the ranking process of retrieved documents that makes the information retrieval more facile.

3.4 COMPLEXITY ANALYSIS OF ALGORITHM

This section discusses time and space complexity of the proposed algorithm and data structure. The work has been carried out with different ways of designing the data structure. In the first case, the element in the second level hash structure holds the URL as string as normally any designer proposes. It is found that the space occupied by the URL name will be repeated in many keyword locations, and hence it is preferred to store the URLs in a sorted array and access them by binary search method and use the index of its position for storing the URL information in the second level hash structure. Hence, both time and space complexity of the proposed data structure with URL name and index of the URL names have been studied. The time complexity is defined as the measure of an algorithm’s total time taken for solving the instance. The space complexity is stated as the measure
of an algorithm’s total memory requirements during run time. Here, the complexity analysis has been accomplished with the assumption of four cases.

1. Time complexity when URLs are stored in canonical string form
2. Time complexity when URLs are stored as index into URL name
3. Space complexity when URLs are stored as strings
4. Space Complexity when URLs are stored as index into URL string table

### 3.4.1 Time Complexity when URLs are Stored in Canonical String Form

Time complexity of ‘BuildKeyDB’ algorithm is carried out as follows:

\[
T_{\text{build}} = HT_1 + HT_2
\]  

(3.2)

where, \( HT_1 \) is the hashing time for first level hash table and it is a function of the length of the keyword, which is given below.

\[
HT_1 = f(k_i)
\]  

(3.3)

\( HT_2 \) is the hashing time for second level hash table and it is a function of the length of the URL.

\[
HT_2 = f(\text{URL}_i)
\]  

(3.4)

Time complexity of sorting the URL table of the concerned keyword in the hash table is carried out by,
\[ T_{\text{sort}} = HT_1 + c_1 n \log_2 n \]  

(3.5)

where, \( n \) is the number of URLs holding the keyword \( K \), and \( c_1 \) is the constant decided by the length of the URL string. Time complexity of searching time for the given keyword is given by \( HT_1 \).

3.4.2 Time Complexity when URLs are stored as index into URL name

The time complexity of ‘BuildKeyDB’ when URLs are stored as index into the URL string table is computed as follows.

\[ T_{\text{build1}} = \log_2 N + HT_1 + HT_2 \]  

(3.6)

Time complexity of sorting the hash table element is

\[ T_{\text{sort1}} = HT_1 + n \log_2 n \]  

(3.7)

where \( n \) is the number of URLs holding the keyword \( K \). Time complexity of searching time for the given keyword is given by \( HT_1 \). Since the length of URL will be around 15 to 50 characters, the space requirement for storing the URL along with the frequency for an associated keyword in the hash element is always higher than that of the space requirements for the URL stored and indexed into the URL table.

3.4.3 Space Complexity when URLs are stored as strings

In a case of memory space requirement when the URLs are stored as string is calculated by the following equations.

\[ S = S_{h1} + S_{h2} \]  

(3.8)
Here, $S_{h1}$ and $S_{h2}$ are the space (memory) requirements of the two hash structures. $S_k$ is the space for the keyword which depends on the length of the keyword. $S_{h2}$ is the memory space requirement of second level hash table for a keyword $K$ and is represented by,

$$S_{h2} = N_{URL,K} \times (URL + sizeof (int))$$

(3.10)

where ‘$n$’ is the number of keywords, $N_{URL,K}$ is the number of documents having the keyword $K$ and $URL$ is the space for URL name.

$$S_{h2} = S_k + 2 \times sizeof (int) \times N_{URL,K}$$

(3.11)

### 3.4.4 Space Complexity when URLs are stored as index into URL string table

The computations of space complexity when the URLs are stored as index with their corresponding keywords are given as follows.

$$S = M \times URL + (S_k + S_{h2}) \times n$$

(3.12)

$$S_{h2} = N_{URL,K} \times (2 \times sizeof (int))$$

(3.13)

where, ‘$M$’ is the corpus size.

### 3.5 RESULTS AND DISCUSSION

The various datasets that are collected from web by using web crawler to produce KIM corpus, 20 Newsgroup dataset, 4 universities dataset and 7 sectors dataset from www.ontotext.com, www.people.csail.mit.edu, and
www.cs.cmu.edu are used in our experiments. Then, some websites are randomly selected to form corpus $C$ (web pages) which consists of 2000 web documents. The efficiency of the proposed work is analyzed with the parameters such as time and space.

The retrieved result for the query word “Linux” is shown in Table 3.1. Column 2 shows the indexing of the web documents based on frequency of terms of appearance in the documents, whereas the column 1 denotes its corresponding URLs. The indexing is made with the frequency impacts of the query term “Linux” over the web documents present in the URLs.

<table>
<thead>
<tr>
<th>URL</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>G:\web1232\ATA-RAID-HOWTO.html</td>
<td>55</td>
</tr>
<tr>
<td>G:\web1232\Accessibility-HOWTO.html</td>
<td>52</td>
</tr>
<tr>
<td>G:\web1232\Astronomy-HOWTO.html</td>
<td>26</td>
</tr>
<tr>
<td>G:\web1232\ADSL-Bandwidth-Management-HOWTO.html</td>
<td>20</td>
</tr>
<tr>
<td>G:\web1232\4mb-Laptops.html</td>
<td>16</td>
</tr>
<tr>
<td>G:\web1232\Accessibility-Dev-HOWTO.html</td>
<td>5</td>
</tr>
<tr>
<td>G:\web1232\3D-Modeling.html</td>
<td>4</td>
</tr>
<tr>
<td>G:\web1232\3-Button-Mouse.html</td>
<td>2</td>
</tr>
<tr>
<td>G:\web1232\Apache+SSL+PHP+fp.html</td>
<td>2</td>
</tr>
<tr>
<td>G:\web1232\AThlon-Powersaving HOWTO.html</td>
<td>2</td>
</tr>
<tr>
<td>G:\web1232\Apache-Overview.HOWTO.html</td>
<td>2</td>
</tr>
<tr>
<td>G:\web1232\Apache-WebDAV-LDAP-HOWTO.html</td>
<td>1</td>
</tr>
</tbody>
</table>

Similarly, many words are given as query terms, and relevant URLs are retrieved and displayed. The offline performance factors such as hash building time, sorting time and online performance factors such as minimum,
average and maximum retrieval time of the query run of complete keywords retrieved from the web pages are tabulated in Table 3.2 and Table 3.3. Table 3.2 shows the performance when the hash elements are organized as URL names and Table 3.3 shows the performance when the hash elements are organized as index to URL names.

**Table 3.2 Performances of the retrieval results with URL in the hash element (all timings in milliseconds)**

<table>
<thead>
<tr>
<th></th>
<th>Corpus 1</th>
<th>Corpus 2</th>
<th>Corpus 3</th>
<th>Corpus 4</th>
<th>Corpus 5</th>
<th>Corpus 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of files</td>
<td>100</td>
<td>250</td>
<td>500</td>
<td>750</td>
<td>1000</td>
<td>1232</td>
</tr>
<tr>
<td>Total words</td>
<td>87962</td>
<td>195300</td>
<td>325000</td>
<td>455584</td>
<td>601849</td>
<td>728842</td>
</tr>
<tr>
<td>Unique words</td>
<td>17435</td>
<td>33479</td>
<td>51864</td>
<td>67765</td>
<td>82444</td>
<td>98339</td>
</tr>
<tr>
<td>Hash Build Time</td>
<td>6047</td>
<td>13047</td>
<td>21157</td>
<td>30922</td>
<td>37093</td>
<td>40968</td>
</tr>
<tr>
<td>Sorting Time</td>
<td>63</td>
<td>125</td>
<td>281</td>
<td>344</td>
<td>469</td>
<td>766</td>
</tr>
<tr>
<td>Minimum Access Time</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Average Search Time</td>
<td>0.0</td>
<td>1.6E-4</td>
<td>1.6E-4</td>
<td>1.6E-4</td>
<td>1.6E-4</td>
<td>3.1E-4</td>
</tr>
<tr>
<td>Maximum Access Time</td>
<td>0</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Total Access Time</td>
<td>0</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>31</td>
</tr>
</tbody>
</table>

Moreover, the table reveals the results corresponding to the six corpus data. The performance analysis has been made with the aspects such as number of files analyzed, total relevant words, unique words, hash build time, sorting time, minimum access time, average search time, maximum access time and total access time. With the same criterions, the performance of the retrieval results with index to the URL in the hash element is examined and the results are displayed in Table 3.3.
Table 3.3 Performances of the retrieval results with index to the URL in the hash element (all timings in milliseconds)

<table>
<thead>
<tr>
<th>No. of files</th>
<th>Corpus 1</th>
<th>Corpus 2</th>
<th>Corpus 3</th>
<th>Corpus 4</th>
<th>Corpus 5</th>
<th>Corpus 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>250</td>
<td>500</td>
<td>750</td>
<td>1000</td>
<td>1232</td>
<td></td>
</tr>
<tr>
<td>Total words</td>
<td>87962</td>
<td>18530</td>
<td>32500</td>
<td>45558</td>
<td>60184</td>
<td>72884</td>
</tr>
<tr>
<td>Unique Words</td>
<td>17435</td>
<td>33479</td>
<td>51864</td>
<td>67765</td>
<td>82444</td>
<td>98339</td>
</tr>
<tr>
<td>Hash Build Time</td>
<td>6016</td>
<td>11203</td>
<td>18297</td>
<td>25500</td>
<td>36140</td>
<td>37726</td>
</tr>
<tr>
<td>Sorting Time</td>
<td>156</td>
<td>187</td>
<td>188</td>
<td>375</td>
<td>328</td>
<td>609</td>
</tr>
<tr>
<td>Minimum Access Time</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Average Search Time</td>
<td>0.0</td>
<td>1.6E-4</td>
<td>3.75E-5</td>
<td>4.0E-5</td>
<td>3.1E-4</td>
<td>4.0E-4</td>
</tr>
<tr>
<td>Maximum Access Time</td>
<td>0</td>
<td>16</td>
<td>15</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Total Access Time</td>
<td>0</td>
<td>16</td>
<td>15</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
</tbody>
</table>

Following the time complexity analysis, the examination is made with the space requirements. Table 3.4 compares the additional memory space requirements of the hash table. The first row comprises the space occupied for storing the organized elements, when the hash table’s elements are organized as URL names, whereas the second row comprises the memory space occupied for storing the index to URL names. Finally, the third row shows the memory space gain corresponding to the six corpuses.

Table 3.4 Memory usage with URL and index to URL in hash element
The efficiency of multilevel hashing mechanism has been effectively enforced in the proposed work and analyzed. Search engines on receiving a request will take a long time to locate required related information for a particular query. Therefore, maintaining all information on the Internet becomes a significant duty to a search engine. With that concern, the work reduces the time taken for retrieving appropriate documents for the given query by using the multilevel hashing data structure.

3.6 SUMMARY

Rate of document available in the internet and other web-oriented services is increasing exponentially. Search engines on receiving a request will take a long time to locate the location of required related information for a particular query. Therefore, maintaining all information on the Internet becomes a daunting duty to the search engine. In order to trim down searching time for location of relevant data, this work proposes a multilevel hashing data structure to arrange the web documents in a database and hash-based on frequency of keywords. Complexity analysis of the algorithm in Section 3.4 and results and discussion in Section 3.5 show that the proposed multilevel hashing data structure retrieved documents as quick as possible. In the next chapter, the concepts are indexed with a similar data structure leading to a fast semantic web access.