DOCUMENT RETRIEVAL BY USING PARALLEL WAVELET TREE

By

Shukrant Tyagi (1402710163)
Sumit Baudh (1402710171)
Vaibhav Kaushik (1402710179)
Vikrant Tyagi (1402710182)

Submitted to the Department of Computer Science & Engineering

in partial fulfillment of the requirements

for the degree of

Bachelor of Technology
In
Conputer Science & Engineering

Ajay Kumar Garg Engineering College, Ghaziabad
Dr. APJ Abdul Kalam Technical University
April 2018

TABLE OF CONTENTS

ABSTRACT……………………………………………………………………………….iv

DECLERATION…………………………………………………………………..v

CERTIFICATE………………………………………………………………………….vi

ACKNOWLEDGEMENTS…………………………………………………………….vii

LIST OF TABLES…………………………………………………………………..viii

LIST OF FIGURES…………………………………………………………………… ix

1 CHAPTER 1 (INTRODUCTION)……………………………………………………..1
- 1.1 INTRODUCTION TO SEARCH ENGINES…………………………………..1
  - 1.1.1 WORKING OF SEARCH ENGINE……………………………………2
  - 1.1.2 COMPONENTS OF SEARCH ENGINE……………………………….5
  - 1.1.3 ADVANTAGES OF SEARCH ENGINE……………………………10
- 1.2 MAJOR OBJECTIVES OF SEARCH ENGINE………………..…………..……10

2 CHAPTER-2 (INFORMATION RETRIEVAL)………………………………………...11
- 2.1 INTRODUCTION……………………………………………………………11
- 2.2 HISTORY……………………………………………………………………..11
- 2.3 COMPONENTS OF INFORMATION RETRIEVAL…………………………..12
  - 2.3.1 WEB CRAWLING…………………………………………………..12
  - 2.3.2 INDEXING……………………………………………….............13
  - 2.3.3 SEARCHING……………………………………………...........13
- 2.4 INDEXING………………………………………………………………...13
  - 2.4.1 TYPES OF TEXTUAL INDEXING………………………...........14
  - 2.4.2 INVERTED INDICES……………………………………….....18
  - 2.4.3 DATA STRUCTURES FOR INDEXING…………………………...27
  - 2.4.4 WAVELET TREE……………………………………………..30

3 CHAPTER-3 (RELATED WORK)…………………………………………........35
- 3.1 INTRODUCTION………………………………………….....................35
- 3.2 SEARCHIN………………………………………...............................35
- 3.3 INDEXING……………………………………….............................36
- 3.4 RANKING………………………………………..............................36

4 CHAPTER-4 (PROPOSED WORK)…………………………………………………38
- 4.1 PROBLEM DEFINITION………………………………………….......38
- 4.2 OBJECTIVE……………………………………….........................38
- 4.3 WAVELET TREE BASED INDEXING……………………….....................39
  - 4.3.1 WAVELET TREE……………………......................................39
  - 4.3.2 INVERTED INDEXING……………………….............................39
    - 4.3.2.1 MAPPER……………………..40
    - 4.3.2.2 REDUCER……………………......................................41

5 CHAPTER-5(ENVIRONMENTAL SETUP)……………………........................48

5.1 DEVCPP…………………………...............................……...........................48
5.1.1 DEVPACKS…………………………………………………….......48
5.1.2 DEVELOPMENT STATUS……………………….......................48
5.2 OPENMP…………………………...............................……...........................49
5.2.1 DESIGN…………………………………….…………………………49
5.2.2 HISTORY……………………………………………………………...50
5.2.3 CORE ELEMENTS………………………………………………51
5.2.4 PERFORMANCE EXPECTATION………………………….........53
6 CHAPTER-6 (EVALUATION STUDYCHAPTER……………………………………54
7 CHAPTER-7 (CONCLUSION AND FUTURE WORKS)……………………….........57
8 REFERENCES...68

ABSTRACT

These days search engines are very popular as we can access the large amount of information available on the World Wide Web. So there is a need of efficient text indexing method. Search engines are used for document and geographical location retrieval, they use inverted indexes for this. In the past the Signature files and the Inverted files were used as indexing techniques. In comparison to inverted files, Signature files require larger space and are expensive in construction. Use of sorted array although guarantees faster access but less efficient updation when compared to B-tree. So, these indexes can be created using different data structures like B tree (for textual indexing), B+ tree (for textual indexing) and wavelet tress. Moreover wavelet trees are efficient data structures as they take less amount of memory storage for storing the inverted index, but the time taken for creation of inverted index is more in wavelet trees as compared to other data structures like B/B+ trees. So, in this paper we are presenting a parallel algorithm to create inverted index using wavelet tree which will take less time as compared to the present sequential algorithm being used for this purpose.

DECLARATION

We hereby declare that this submission is our own work and that, to the best of our knowledge and belief, it contains no material previously published or written by another person nor the material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgement has been made in the text.

Signature :

Name : Shukrant Tyagi

Roll No : 1402710163

Date :

Signature :

Name : Sumit Baudh

Roll No : 1402710171

Date :

Signature :

Name : Vaibhav Kaushik

Roll No : 1402710179

Date :

Signature :

Name : Vikrant tyagi

Roll No : 1402710182

Date :

CERTIFICATE

This is to certify that Project Report entitled "Document Retrieval By using Parallel Wavelett Tree " which is submitted by Shukrant tyagi(1402710163), Sumit Baudh(1402710171), Vaibhav Kaushik(1402710179), Vikrant tyagi(1402710182) in partial fulfillment of the requirement for the award of degree B.Tech. in Department of Computer Science and Engineering of Dr. APJ Abdul Kalam Technical University, is a record of the candidate's own work carried out by them under my supervision. The matter embodied in this thesis is original and has not been submitted for the award of any other degree.

Date **Supervisor**

Dr. Arun Kumar Yadav

(Assistant Professor)

CSE Department

ACKNOWLEDGEMT

It gives us a great sense of pleasure to present the report of the B.Tech Project undertaken during B.Tech Final Year . We owe special dept of gratitude to Dr. Arun Kumar Yadav, Department of Computer Science & Engineering, Ajay Kumar Garg Engineering College, Ghaziabad for his constant support and guidance throughout the course of our work. His sincerity, thoroughness and and perseverance have been a constant source of inspiration for us. It is only his cognizant efforts that our endeavors have seen light of the day.
We also take the opportunity to acknowledge the contribution of Dr. Sunita Yadav, Head, Department of Computer Science & Engineering, Ajay Kumar Garg Engineering College, Ghaziabad for his full support and assistance during the development of the project.
We also do not like to miss the opportunity to acknowledge the contribution of all faculty and staff members of the department for their kind assistance and cooperation during the development of our project. Last but not the least, we acknowledge our friends for their contributions in the completion of the project.

Signature	:
Name	: Shukrant Tyagi
Roll No	: 1402710163
Date	:
Signature	:
Name	: Sumit Baudh
Roll No	: 1402710171
Date	:
Signature	:
Name	: Vaibhav Kaushik
Roll No	: 1402710179
Date	:
Signature	:
Name	: Vikrant tyagi
Roll No	: 1402710182
Date	:

LIST OF TABLES

Table No.	Table Name	Page No.
2.1	Document Table	15
2.2	Bitvector Table	15
2.3	Input Array	16
2.4	Resultant Array	16
2.5	Input Data	18
2.6	Inverted Index	19
2.7	Inverted Index with More information	19
2.8	Document List	21
2.9	Mapping of terms to their hash values	21
2.10	Mapping of terms to their hash value with more information	22
2.11	Document	23
2.12	List	24
2.13	Sorted List	24
2.14	Dictionary and posting file from the last example	25
2.15	Document	26
2.16	Inverted Index	26
2.17	Inverted File	33
6.1	Time of Algorithm	54

LIST OF FIGURES

Figure No. Page No.

Fig 1.1 Data Structure for Web Document in Repository----------------------------------3

Fig 1.2 Showing flowchart of working of search engine------------------------------------4

Fig 1.3 Data Structure for Web Document in Repository----------------------------------5

Fig 1.4 General Architecture of search engine--7

Fig 1.5 Indexing process---8

Fig 2.1 Architecture of Web Crawler--12

Fig 2.2 Complex inverted index--20

Fig 2.3 Overall schematic of sorted array inverted file creation---------------------------23

Fig 2.4 B-Tree---27

Fig 2.5 An inverted index--28

Fig 2.6 Fixed Length code ---29

Fig 2.7 Sorting suffix alphabetically---30

Fig 2.8 Binary Search---30

Fig 2.9 Searching of sequences start with ‘at’--30

Fig 2.10 Wavelet Tree---33

Fig 4.1 Unique terms with list of document id’s of documents-----------------------------39

Fig 4.2 Posting List---40

Fig 4.3 Working of Mapper--40

Fig 4.4 Construction of Wavelet Tree for string--42

Fig 4.5 Screenshot of Result---47

Fig 5.1 An illustration of multithreading--50

Fig 5.2 OpenMP Construct --53

Fig 6.1 Words VS Time Graph(Intel i3 Processor)--55

Fig 6.2 Words VS Time Graph (Intel i5 Processor)---56

Fig 6.3 Words VS Time Graph (Intel i7 Processor)---56

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION TO SEARCH ENGINES

A web search engine is a software system that is designed to search for information on World Wide Web t indexes millions of web pages, answers for millions of queries. There are three important work for any search engine by which it differ in various ways

- Perform searching on the internet based on important words
- Indexing of searched words
- Permit user to search words or their combination

Primary goal of search engine is to provide relevant results to the user over a rapidly growing world wide web. The most important measure for search engine is indexing of web documents efficiently for increase search performance and quality of results. It is clear from above discussion that web directories are not sufficient for searching due to huge amount of data. Several search engines are developed from 1994 like Lycos, Ultra vista, Excite, Infoseek, Generally search engines collect web pages in their own repository then index these documents for later retrieval of information. Internet is nothing without these search engines. GUI, Repository, Index, Crawler are the major and important components of the search engine. User provide query to search engine's interface then search engine finds matching documents from its database and show results after ranking of these documents

Brin defines the Web as follows (1998): The Web is a vast collection of completely uncontrolled heterogeneous documents", The Web is accessible to anyone via a Web browser. Search engines answer tens of millions of queries every[1]. The amount of information on the Web grows exponentially. If we only count the textual data, it is estimated to be in the order of one terabyte[2). Of course, the size of multimedia data types (image, audio, and video) is even bigger. Since this data is managed by many individuals, organizations, and companies, thus, the World Wide Web can be seen as a large unstructured and ubiquitous database. Finding useful information on the Web is a very difficult task. There is a need to develop tools to manage, retrieve, and filter information in the big database)

In fact Internet would have not become so popular if Search engines would not have been developed. User interface, query engine, indexer, crawlers) and repository are the major components of a Search engine. To access information from the www, users provide search queries in the search engine's interface. In response to the search query provided Search engines use their database to search the relevant documents and produce the resultafter ranking it on the basis of relevance. Though all components of Search engines playmajor role, the quality of search results depends on the efficiency of its database repository In fact, the search engine builds its database, with the help of crawlers, where a crawler is program that traverses the Web and collect information about web documents. It extracts the links from the collected pages and follows them to download related pages and so on Starting in 1994, a number of search engines were launched, including Alta Vista, Excite Infoseek, Inktomi, Lycos, and of course the evergreen, Yahoo and Google. Most of these search engines save a copy of the web pages in their central repository and then make appropriate indexes of them for later search/retrieval of information [4]

The search engine works in the following way: Firstly, the URL server gives the list of seed URLs to the web crawler. Then fetched web pages are sent to the store server. Store server stores and compresses the web pages into repository. Every web page has an associated IDnumber which is called as doe ID. Indexing function is performed by indexer and sorted Indexer performs several functions like read the repositories, uncompressed web documents and parse them. Every web document is converted into set of word occurrences called hits Indexer distributes these hits into barrels and form forward index. Indexer also parses all the links and store information about these links into the anchor files. URL resolver reads anchor files and converts relative URLs to absolute URLs. It provides anchor text to forward index, associated with doc ID that the anchor points to. It also generates pairs of doc IDs.The link database is used to compute page ranks for all documents. Sorter focuses on barrels and form inverted index from forward index. It also generates a list of word IDs and offset into inverted index. A program Dump lexicon takes this list and generate new lexicon used by searcher. Web server runs searcher and page ranks to answer the queries.

Meta search -These are different type of search engines because they do not have their own repository index. Although there is one GUI (Graphic User Interface). When user provide the query to the GUI then query goes to different search engines and results come from different search engines and these results are compiled after removal of duplication Results displayed to the user after ranking process. Some examples are like Dog pile, Surfwax,37.com.

1.1.1 WORKING OF SEARCH ENGINE

A search engine finds information for its database by accepting listings sent in by authors who want exposure, or by getting the information from their web crawlers," "spiders," or "robots, programs that roam the Internet storing links to and information about each page they visit the figure 1.2 shows basic flow diagram of search engine. A web crawler is program that downloads and stores Web pages, often for a Web search engine. Roughly, crawler starts off by placing an initial set of URLs, in a queue, where all URLs to be retrieved are kept and prioritized. From this queue, the crawler gets a URL (in some order), downloads the page, extracts any URLs in the downloaded page, and puts the new URLs in the queue. This process is repeated until the crawler decides to stop. Collected pages are later used for other applications such as web search engine or a web cache[5]

Search engines are another type of information retrieval tools used to access information from the Internet. Starting in 1994, a number of search engines were launched, including Alta Vista

Excite, Infoseek, Inktomi, Lycos, and of course the evergreen, Yahoo and Google. Most of these search engines save a copy of the web pages in their central repository and then make appropriate indexes of them for later search /retrieval of information.

The search engine works in the following way: Firstly, the URL server gives the list of seed URLs to the web crawler. Then fetched web pages are sent to the store server. Store server stores and compresses the web pages into repository. Every web page has an associated ID number which is called as doc ID. Indexing function is performed by indexer and sorter, Indexer performs several functions like read the repository, uncompressed web documents and parse them. Every web document is converted into set of word occurrences called as hits. Indexer distributes these hits into barrels and form forward index. Indexer also parses all the links and store information about these links into the anchor files. URL resolver reads anchor files and converts relative URLs to absolute URLs. It provides anchor text to forward index, associated with doc ID that the anchor points to li also generates pairs of doc IDs. The link database is used to compute page ranks for all documents. Sorter focuses on barrels and form inverted index from forward index. It also generates a list of word IDs and offers into inverted index. A program Dump lexicon take this list and generate new lexicon used by searcher. Web server runs searcher and page ranks to answer the queries.

The basic techniques for any search engine to function are crawling, indexing and ranking, Crawler is a program used to find the information on the web. It works like a spider to search web documents and it is also called as robot. Crawler automatically downloads the web pages for indexing and follow the links of web pages. Some modules as shown in figure are URL resolver 6), store server etc.

Some major data structures search engine (Brin et al., 1998) are:

Repository

A web document repository contains web pages one after the other. All these web pages are integrated with document ID, length of URL. Data structure for this repository(figure 1.1) is given as (Brin et al., 1998)

Doc ID	EnCode	URL Length	Page Length	URL of Page

Fig. 1.1 Data Structure for web document in a repository

Document Index

It keeps all information about each web page which is ordered by document ID. Each entryincludes checksum, doc status, pointer into repository, and other statistics.

Indexer

It is a program that reads the web pages which are downloaded by crawlers. Indexer examines the HTML code and look for terms that are more important. Every web database has different types of indexing

Hit List

It is a list of terms and their occurrences in particular document. It accounts for most of the space in both forward and inverted indexing so it is necessary to represent it efficiently.

The flowchart of the essential steps involved in the working of a search engine is in figure 1.2

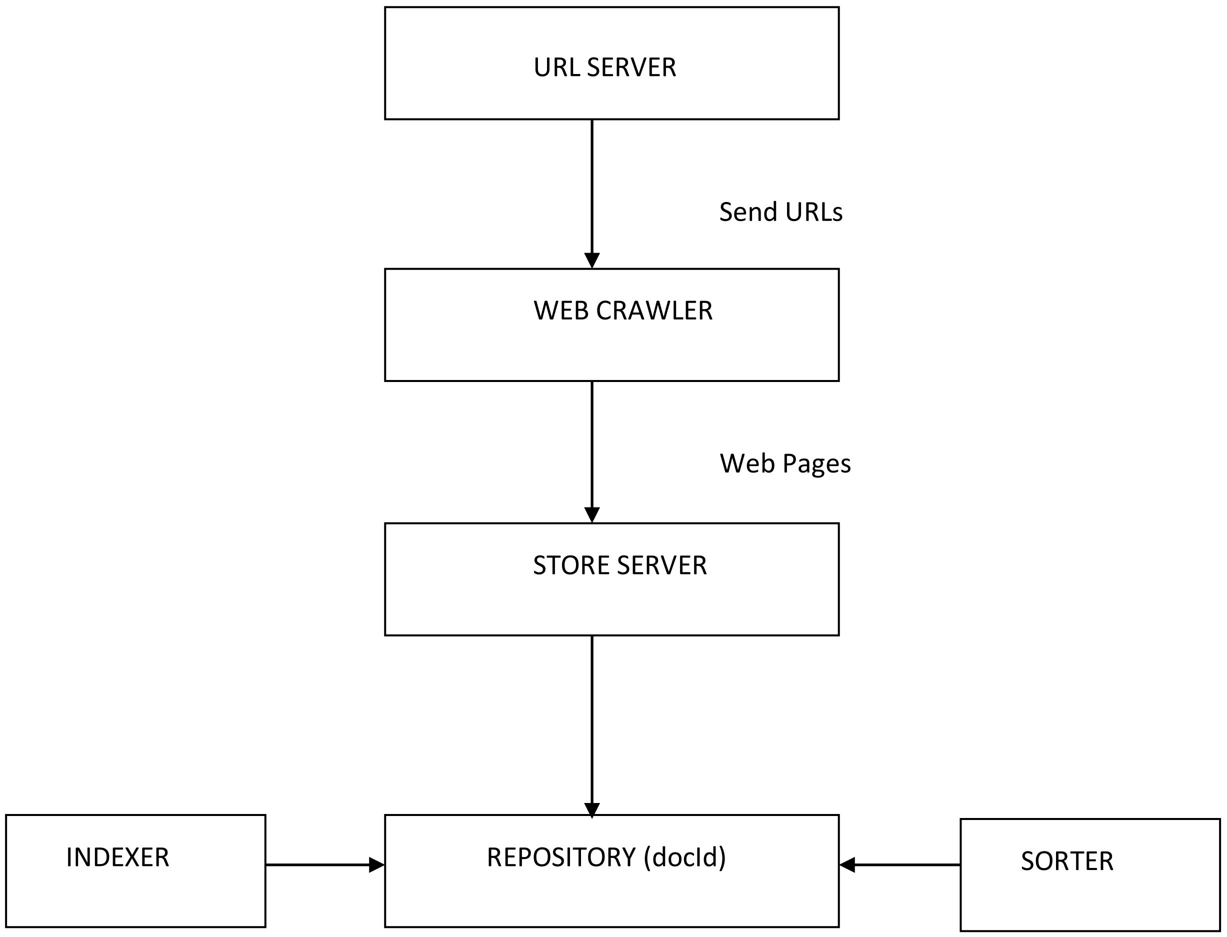


Figure 1.2 Showing flowchart of working of search engine.

There are some problems when users use the interface of a search engine-

- The users do not exactly understand how to provide a sequence of words for the search
- The users may get unexpected answers because he/she is not aware of the input requirement of the search engine. For example, some search engines are case sensitive

- The users have problems understanding Boolean logic: therefore, the user cannot perform advanced searching.
- Novice users do not know how to start using a search engine.
- The users do not care about advertisements, so the search engine lacks funding.
- Around 85% of users only look at the first page of the result, so relevant answers might be skipped.

In order to solve the problems above, the search engine must be easy to use and provide relevant answers to the query [7]

The Google search engine (www.google.com) heavily uses the structure present in hypertext. It claims that it produces better results than other search engines today it references about 24 million pages Google is mainly written in C/C++ for efficiency reasons. It can run on Solaris or Linux platforms. The architecture is shown in the Figure 1.4

1.1.2 COMPONENTS OF SEARCH ENGINE

Some major components of search engine are [9]:

Crawlers

A crawler is a program which visits every page and its every connected page on every site that is searchable or that wants to be searched and reads it. It uses the hypertext links for finding the paths over net and to discover and read other pages of a site Thus is given the same crawler because it crawls each and every page over the net. Thus a crawler can be described simply as a program that automatically retrieves documents according to the user's need starts crawling of web documents from the collection of URLs called as seed documents and find the hyperlinks of web documents. Hyperlinks form a graph like structure. Thus it traverses in a graph by using some graph traversing strategies such as BFS (Breadth first search and DFS (Depth first search. Several policies are used here for crawling purpose and these are selection policy revisited policy and politeness policy.

Repository

A web document repository (fig. 1.3)contains web pages one after the other. All these web pages are integrated with document ID, length of URL etc. The data structure for repository is given as

Doc ID	EnCode	URL Length	Page Length	URL of Page

Figure 1.3 Data Structure for a Web Document in a Repository

Query Engine

The main objective of a search engine is to provide a good quality search result against the user query. The query engine uses some algorithm to perform this task, the query engine calculate the relevance of matched documents, with the help of ranking found for queries provided by the users.

Ranking

Ranking is very interesting feature of the search engine. the results that are searched for user query may be 1000s in number so the ranking is a way with which the pages are prioritized such that whenever asked the are shown in the order of their relevancy such that the most relevant results is shown first.

Generally relevance is measured on the basis of thematic and geographic similarity but thereare many other factors which affect ranking of the document. Let us take an example "monuments near Yamuna", nearby monuments would be ranked high. User probably alsowants some other monuments which are farther away, especially when web documents related to these monuments are more relevant thus the ranking mechanism should be an ideal way to combine the content and geographic relevance of web documents with respect To user query to take care of relevance, redundancy and novelty. Some other factors could be there which are independent of query but significant to provide results. Thus a categorization of these parameters is presented here:

1. Thematic relevance

It is used to measure aboutness of documents with respect to query topic, or matching between textual content of documents and query.

2. Location relevance

This is related to geographic dimensions of relevance. Relevancy is measured on the basis of spatial relationships between documents and queries in geographical space.

3. User centered contextual relevance

Current geographic information retrieval ranking methods are mainly concerned with retrieving documents on the basis of location. But there are many contextual factors that could influence the importance of the web page for spatial information need.

4. Query independent spatial relevance

Some query independent relevance measures such as link structure of the web can be used for ranking purpose. This method takes geospatial properties of the web pages into account should provide a better measure of geospatial importance

The Figure-1.4 of the essential steps involved in the working of a search engine is as follows:

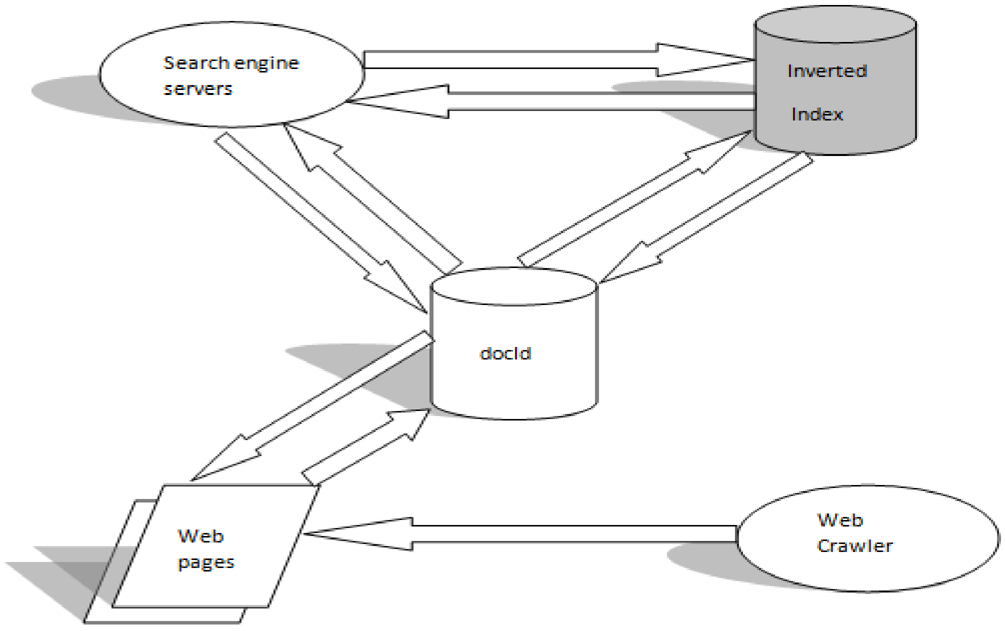


Figure 1.4 General Architecture of search engine

Indexing

To identify the theme result from huge database, indexing plays important role to search relevant data in minimum time. Indexing should also take memory space in to the consideration and should be done in such a way so that it utilizes the space in an efficient manner

Indexing is defined as a data structure technique which facilitates efficient retrieval of records from the database files based on some attributes on which the indexing has been done[10). Thus, Indexing is a process to create tables (indexes) that point to the location of folders, files and records. It helps in recognizing the location of resources depending on file names, key data fields in a database record, text within a file or unique attributes in graphics or video file (1) Indexing can also be defined as the transformation of documents to searchable data structures. It may be manual or automatic

An index creates basis for direct search or for search through indexed 12). The designing of Index consists of interdisciplinary concepts from linguistics, cognitive psychology. mathematics, informatics, and computer science Database indexing is a data structure technique which makes the data retrieval operations on a database table faster with little overhead of additional writes and the storage required to maintain the index[12] Figure 1.5 shows the basis structure index creation and it's working

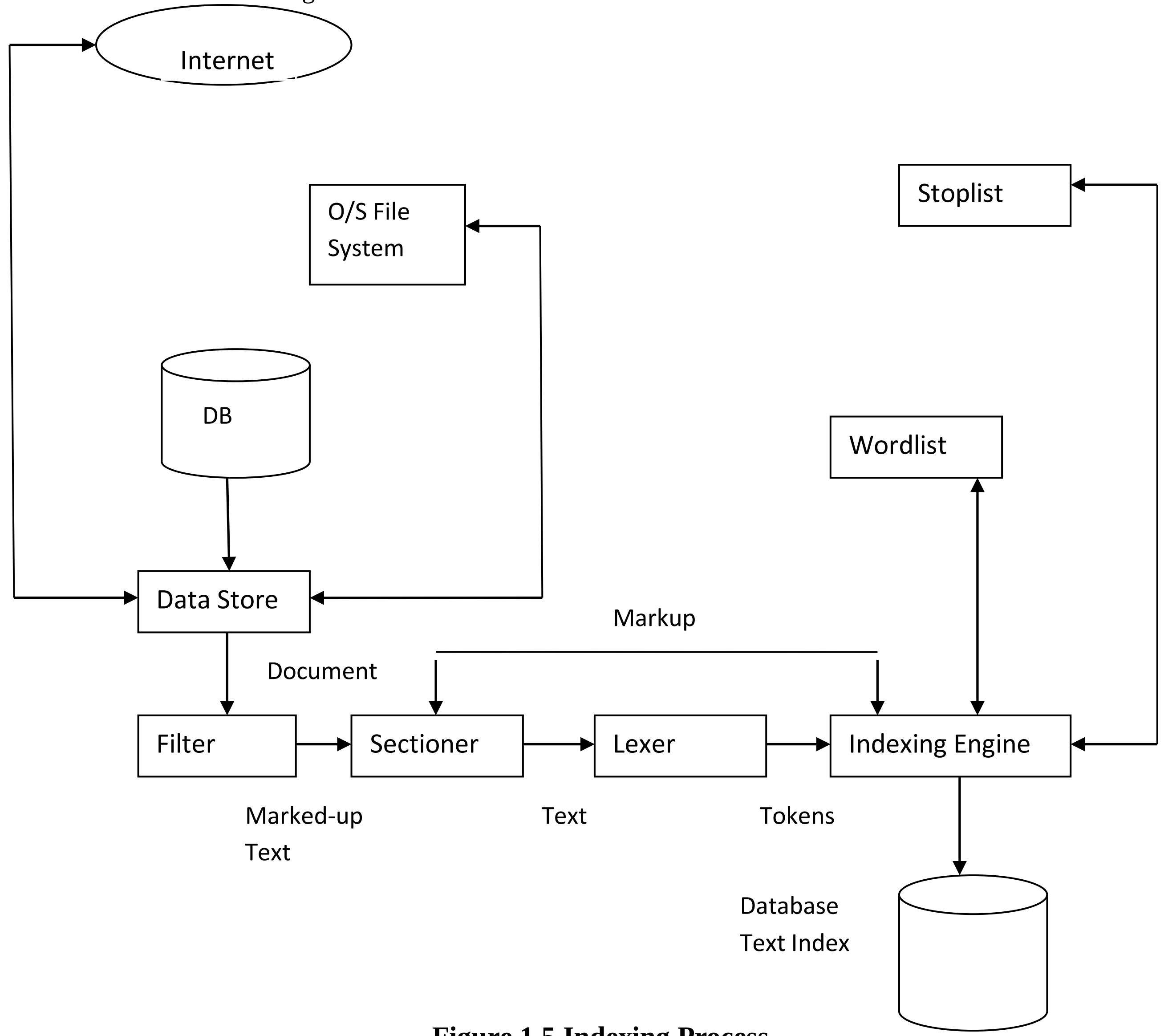


Figure 1.5 Indexing Process

The prior to the process of indexing the documents to be indeed are extracted from the internet databases this huge collection of documents works as the data-store to the indexing process. Further this data is filtered for the replication. So we get a set of non-replicated set of documents these documents are generally in the form of marked up text the sectionedseparates the mark up tags from the text in the documents. The lexer user this long text to convert it into smaller unit words called the tokens. the indexing engine uses certain set of algorithms to use these tokens and wordlist and stoplist to construct the index.stopword are the words that are commonly not added to the index. The constructed index table is stored in the database with a suitable data structure for faster and efficient retrieval construction and modification

Major Objectives of Indexing

Indexing help by providing electronic access points to collection, and the ability of being able to search across resources more effectively. The major Objectives of Indexing can be listed as

1. Fast Searching and Retrieval[13]

The aim of storing an index is to optimize speed and performance in searching required information for a search query. It helps searching by building a list of all of the files located on system for faster queries.

2. No duplicated data storage

Any information should only be indexed once. There should be no duplicated data storage at any index. This not only saves the storage space but also reduces the searching time for any information. This in tum speeds up retrieval process and also reduces the cost of storage.

3. To indicate right position of files and folder[13)

One of the main objectives of creating an indexed data structure is to indicate the correct location of files and folders. In absence of a relevant index, the search engine would scan each and every file/document in the database, which would then consume considerable time and computing power.

4. Information Collection

Indexes are helpful in collecting information in case of huge data size

5. Maximizing the searching success[14]

Organizing the data in an indexed data structure for information retrieval not only makes the search process faster and easier but also increases the chances of search result more successful Other

6. Objectives

a. Identifying the units of documents that identify particular topics or contain certain features

b. Provision of access through synonyms or other terms that are equivalent to the oneterm indexed.

c. Using the terminologies of prospective users and thus allow access to certain topicsor features

d.Help users by guiding them to related concepts topics of documentary units.

e. Indicating, according to the level of exhaustively, all the important features andor features.

f. Using the concepts based on verbal text being indexed, provision of access to topics

1.1.3 ADVANTAGES OF SEARCH ENGINE

There are three very compelling advantages of most search engines.

The indexes of search engines are usually vast, representing significant portions of the Intemet, offering a wide variety and quantity of information resources.

The growing sophistication of search engine software enables us to precisely describe the information that we seek.

The large number and variety of search engines enriches the Internet, making it at least appear to be organized.

1.2 MAJOR OBJECTIVES OF SEARCH ENGINE

Some objectives of search engine are:

- Search engines must update their repository on a regular basis as millions of new web documents are added day by day due to this it is very much dynamic in nature.
- It should elaborate and download the web pages as much as possible.
- It treats spatial terms as the textual terms so there is no difference between textual and spatial. Search engine should develop both type of indexing i.e., textual indexing and spatial indexing
- It should display relevant results within appropriate time.
- There should be an efficient ranking algorithm.
- As millions of new web documents are added day by day due to this it is very much dynamic in nature so search engines must update their repository frequently
- It should elaborate and download the web pages as much as possible
- It treats spatial terms as the textual terms so there is no difference between textual and spatial. Search engine should develop both type of indexing ie, textual indexing and spatial indexing
- It should display relevant results within appropriate time.
- There should be an efficient ranking algorithm.

Internet would have not become so popular if Search engines would not have been developed. User interface, query engine, indexer, crawler(s) and repository are the major components of a Search engine. To access information from the WWW, users provide search queries in the Search engine's interface. In response to the search query provided, search engines use their database to search the relevant documents and produce the result after ranking it on the basis of relevance. Though all components of Search engines play major role, the quality of search results depends on the efficiency of its database/repository In fact, the Search engine builds its database, with the help of crawlers, where a crawler is program that traverses the Web and collects information about web documents. It extracts the links from the collected pages and follows them to download related pages and so on.

CHAPTER 2

INFORMATION RETRIEVAL

2.1 INTRODUCTION

Information Retrieval (IR) is defined as an activity which deals with obtaining of information resources that are relevant to an information need from a collection of information resources IR process starts when users enter a query into IR System. Queries are defined as formal statements of information need. The IR System then locates information that is relevant to user's query.

2.2 HISTORY

In Pre-History, Mechanical & electro-Mechanical devices were used for IR.[15]The term "Information Retrieval" was coined by Calvin Mooers, while presenting a paper (describing IR systems using punch cards) at a conference in March 1950, in which he wrote "The problem under discussion here is machine searching and retrieval of information from storage according to a specification by subject... It should not be necessary to dwell upon the importance of information retrieval before scientific groups such as this for all of us have known frustration from the operation of our libraries all libraries, without exception [16).

IR was popularized by the article As We May Think by Vannevar Bush in 1945[17] that was inspired by patents for 'statistical machine (that searched for documents stored on filmy filed by Emanuel Goldberg in 1920s and '305[18).A computer searching for information was first described in 1948 by Holmstrom. Automated information retrieval systems were introduced in 1950s.

In 1960s, first large information retrieval research group was formed by Gerard Salton at Comell. Another significant innovation of this time was the relevance feedback introduction 19 i.e. to support iterative search, where documents previously retrieved could be marked as relevant in an IR system. Other IR enhancements of this period included the combining of documents with similar content.

By 1970s several different retrieval techniques had been shown that performed well on small text corpora like the Cranfield collection (several thousand documents). Large scale retrieval systems, like the Lockheed Dialog system, came into use early in the 1970s. One of the most important developments of this period was that Luhn's term frequency (f) weights (based on the occurrence of words within a certain document), were complemented with Sparck Jones's work on word occurrence across the documents of a collection.

In 1992, US Department of Defence along with the National Institute of Standards and Technology (NIST), cosponsored the Text Retrieval Conference (TREC) as part of the TIPSTER text program[20]. Its aim was to examine the information retrieval community by providing the infrastructure required for evaluation of text retrieval methodologies on a very test collection.

2.3 COMPONENTS OF INFORMATION RETRIEVAL

There are mainly three components of information retrieval.

- Web Crawling
- Indexing
- Searching

2.3.1 WEB CRAWLING

According to Wikipedia" A web Crawler is a program or automated script which browses the World Wide Web in a methodical, automated manner. We also known it as web spider or web robot. This is the known as web crawlingThere are many websites, basically in search engines, use web crawling to provide updated data. It downloads copy of all the web pages it visit and process that data that helps in index making in further steps. There are also many other works that a spider do. These are maintenance of a website, checking links and validating HTML code. Other works of web crawling could be harvesting an e-mail[21

The behavior of a Web crawler is the outcome of a combination of policies[22):

- A selection policy that states which pages to download,
- A revisit policy that states when to check for changes to the pages,
- A politeness policy that states how to avoid overloading Web sites, and CA parallelization policy that states how to coordinate distributed web crawlers.

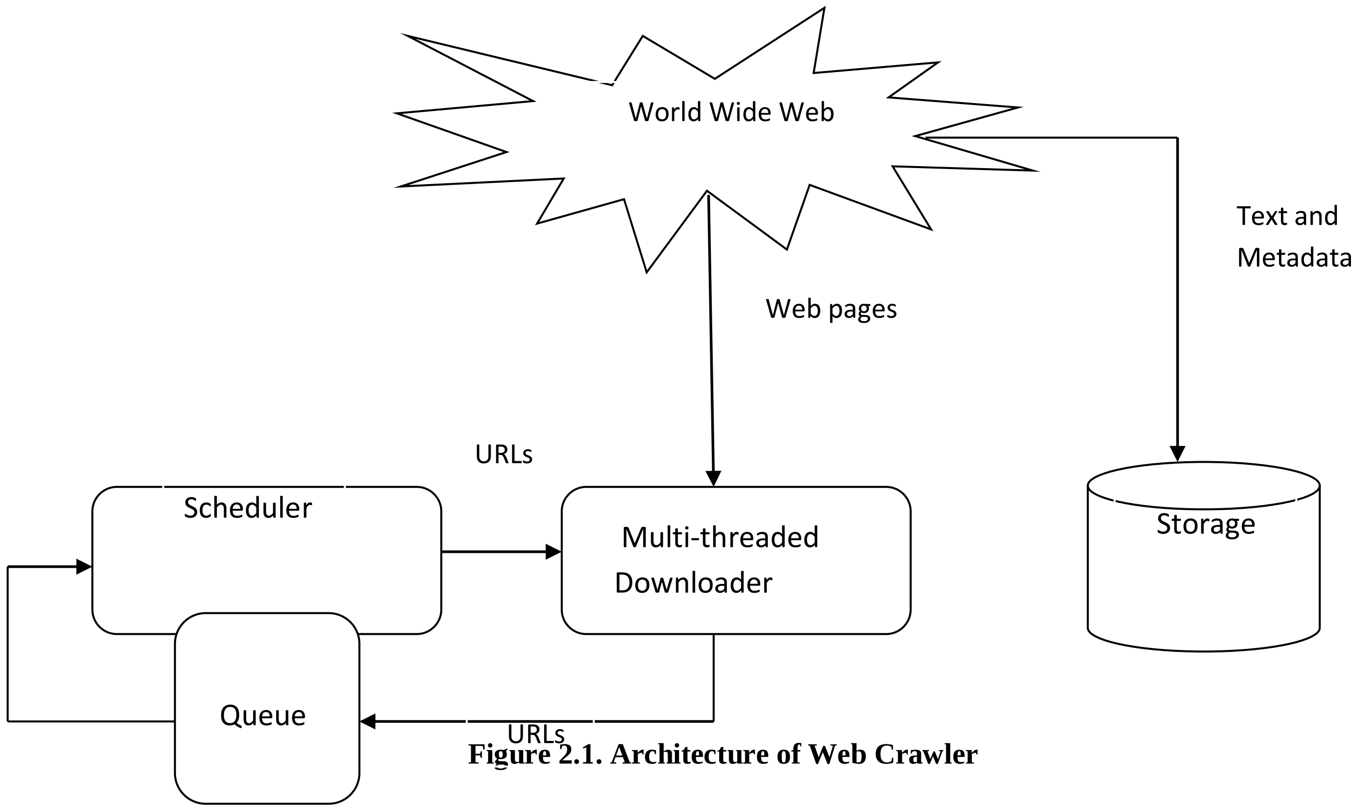


Figure 2.1. Architecture of Web Crawler

During this process it can happen that we cannot he continued from some particular URI. This case in the following cases[22]

- The server containing the URL is taking too long to respond.
- Server is not allowing access to the crawler.
- URL is left out to avoid duplication
- The crawler has specially been designed to ignore such pages. This is done under Politeness Policy of the crawler
- Usually Crawlers crawls the web pages only till 5 depth, it is because of the following reasons
- Normally 5 depth or levels of search is enough to gather the majority of information present in the home page of website

It can be seen as a precaution to avoid "Spider Traps

2.3.2 INDEXING

Search engine indexing collects pares, and stores data to facilitate fast and accurate information retrieval. Index design incorporates interdisciplinary concepts from linguistics, cognitive psychology, mathematics, informatics, and computer science[23]. An alternate name for the process in the context of search engines designed to find web pages on the Internet is web indexing Popular engines focus on the full-text indexing of online, natural language documents

The purpose of storing an index is to optimize speed and performance in finding relevant documents for a search query(22). Without an index, the search engine would scan every document in the corpus, which would require considerable time and computing power. For example, if we have 10.000 documents that can be queried within milliseconds, a sequential scan of every word in these documents could take much longer time[24]. The added computer storage needed to store the index, as well as the amount of time increased is traded off for the time saved during retrieval of the information

2.3.3 SEARCHING

For the user query searching is done such that it produces the most appropriate results for that query[25]. Some search engine uses Boolean logic theory for getting the appropriate results. For the user query most relevant documents are retrieved from the database. The retrieved documents are then ranked according to their relevancy with the user search query the most relevant document webpage is displayed at the top.

These Webpages are ranked by several approaches. Some search engines have algorithms and formulas to do that action and some do this by reading the HTML tags of the Webpages.

In the later terminology, contents of title tag, head tag and footer tag has the highest priority in a particular webpage.

2.4 INDEXING

If a small number of documents needs to be processed then it is possible to use full text search i.e. Serial scanning .But when the large no. of documents are to be processed then indexingis needed to be done in indexing build a list of all the search words thus while searching the of whole documents. The Indexer will create an entry in the index for each found in a document and possibly note its relative position within the document.

High performance text indexes are very important in the use of modern computer Applications from the large web-based search engines to the "find facilities included in popular operating systems, and from digital libraries to online help utilities

Many improvements have been encountered in the efficiency of query evaluation using such indexes. [26]

These advances have been complemented by new methods for building indexes (27).

The suffix tree and suffix arrays method to index the text has been independently discovered[28].

Their study revealed the fact that the suffix arrays are more space efficient than suffix trees but it requires longer time to build it than suffix tree.

An indexing is needed to be done such that CPU cycles and disk cycles would Reduce Or in simple words we can say that indexing is done to reduce the time and space complexity

In this paper we Discuss various index construction strategies However, as a typical document contains hundreds of distinct terms, such an update involves hundreds of disk accesses for this season we need to use an efficient strategy data structure for index construction thus we would suggest a new index construction technique using wavelet tree

Until now, the three main strategies that are used to construct an index are inverted file signature file and bitmap. for an effective information retrieval in keyword based searching and location based searching we need to apply an efficient indexing technique. In our paper we have given brief idea about the other techniques and focus is on inverted file indexing

Inverted files are the only effective structure for supporting text search. An invested indexis a data structure that maps from a tem word), to a list that identifies the all documentscontaining that term. For efficiency at search time, each postings list is stored contiguously

Each postings list in a typical implementation contains the number and locations of tem occurrences in the document, for each document in which the term occurs.

Several reductions can be made More compact alternatives are to omit the locations, or even to omit the number of occurrences, recording only the document identifiers

Although inverted index reduces fastens the query processing time but at the cost of resource intensive task

2.4.1 TYPES OF TEXTUAL INDEXING

The four main index construction techniques are discussed below

Bitmap

A bit map is a special type of inverted files. In A bitmap for every term in the vocabulary bit vector stored, each bit corresponding to a document. A bit is set to 1 if the term appears anywhere in that document and otherwise[29]

Let's see an example.

Document	Text
1	Ram and Shayam are friends
2	Priya and rita are enemy
3	Priya is mother of ram
4	Shayam is father of rita

Table 2.1 Document table

Thus we need to calculate the bit vector for each unique term thus the below example shows the calculation for the bit vectors for term and

A bit vector is presented by bits bl. 2...., bn where in is the no. of documents in database

Thus if a word appears in a document then the corresponding bit is set to 1 else 0

For key word "and" the corresponding bit vector would be 1100 because it appears on document (1.2) and not in document (3,4)

similarly, others can be calculated

Term	Bitvector
Ram	1010
And	1100
Shayam	1001
Are	1100
Friend	1000
Priya	0110
Rita	0101
Enemy	0100
Is	0011
Mother	0010
Father	0001
Of	0011

Table 2.2 Bit vector table

Bitmaps are particularly efficient for answering Boolean queries. Using bitmap makes the search and use faster but at the cost of huge storage needed

Signature files are space efficient and are adjustable reducing the size of a signature file increases the no of fate matches and in other way round retrieval may become faster if index is enlarge

Suffix Array

In paper a data structure is proposed that is simple and efficient for online string searches called the suffix arrays earlier suffix tree are used but this proposed structure of suffix improves the space requirement of suffix tree by three to 5 times 30). A suffix array is very efficient to process queries like finding a substring and that too with a time complexity of O(A log K), where A refers to length of given string W which is needed to be checked and K is the length of parent string has (KLOGK) But The time for construction used in the suffix tree is O(K) in worst case while suffix array is (KLOGK)

For indexing a suffix tree is made as an array of integers consisting of start positions of all the suffixes formed from a string in alphabetical order. If we need to search any string than we use suffix array by applying a search technique on its suffix array

Mostly the suffix array is used for large contextual document.

Index	Suffix of S
1	Rababra$
2	Ababra$
3	Babra$
4	Abra$
5	Bra$
6	Ra$
7	A$
8	$

Sorted Index
2
4
7
3
5
1
6
8

Table 2.3 Input Array

Index	Suffix of S
2	Ababra$
4	Abra$
7	A$
3	Babra$
5	Bra$
1	Rababra$
6	Ra$
8	$

Table 2.4 Resultant array

Suffix array gives best result in the cases like phase searching searching with the help of regular expression, longest repletion and string search

Signature Files

idea behind a signature file is just like a filter that quickly discards many of the non-qualifying items it is to be ensured that the qualifying terms definitely pass the test and may be possible that along with the qualifying term, some other non- relevant terms are also passed can be called the miss or false hits unknowingly

Thus the whole process in includes storing the documents sequentially in a text file, a hash coded bit patterns are created called the signature and stored in the "signature file.". When query arrives, the signature file is scanned and many non qualifying documents are or can be presented to user as it is discarded. The one's not discarded are then either checked, full text search can be applied,

comparative study of signature files with other method of indexing are done in variousresearches shows that the signature-based methods are much faster than full text scanning(1 or 2 orders of magnitude faster, depending on the individual method). Compared to inversion, they require a modest space overhead (typically 10-15%[29], as opposed to 50 -300% that inversion requires moreover, they can handle insertions more easily than inversion, because they need "append-only" operations-o reorganization or rewriting of any portion of the signatures thus concurrency can be increased during insertions and these methods work well on Write-Once-Read-Many (WORM) optical disks, which constitute an excellent archival medium [31]

On the other hand, signature files may be slow for large databases, precisely because their response time is linear on the number of items in the database

Superimposed coding(32) is one of the widely used approaches for creation of signature files in this method document is divided into "logical blocks," that is, pieces of text that contain constant number D of distinct, non -common words. A word signature is then created for cache word, which is a bit pattern of sizes, with b bits set to "1", while the rest are "0" s and The word signatures are OR together to form the block signature. Block signatures are words to the signature of the document than block concatenated, to form the document signature. The b bit positions to be set to "l' by each word are decided by hash functions. Searching is done by comparing the signature of that word to the signature of the document then block.

Thus we need to calculate the signature for each unique term thus the below example shows the calculation for the signature for term ‘and’ .

Binary equivalent of and

a-01100001

n=01101110

d=01100100

let h(x) be the mapping function thus

signature of and h(and)= 011 000110

similarly, others can be calculated

Word Signature

And	011 000 110
Are	011 010 111

Block signature 011 010111

Inverted Files

Vocabulary, a list of all terms that appear in the database, is made. The vocabulary supports mapping from terms to their corresponding inverted lists and in its simplest form is a list of strings and disk addresses

Document	Text
1	Ram and Shayam are friends
2	Priya and rita are enemy
3	Priya is mother of ram
4	Shayam is father of rita

Table 2.5 Input Data

2.4.2 INVERTED INDICES

In document level construction each term is represented by a triplet <[number of occurrences];[documents number separated by comma,”,”];[term]>

Document level construction:

Number	Term	Document
1	Ram	<2,1,3>
2	And	<2,1,2>
3	Shayam	<2,1,4>
4	Are	<2,1,2>
5	Friend	<1,1>
6	Priya	<2,2,3>
7	Rita	<2,2,4>
8	Enemy	<1,2>
9	Is	<2,3,4,>
10	Mother	<1,3>
11	Father	<1,4>
12	Of	<2,3,4>

Table 2.6 Inverted Index

The granularity of an index is the accuracy to which it identifies the location of a term.

So, in order to increase granularity In document level construction each term is represented by a triplet [no. of occurances) [(document no separated by comma; position

Word level Construction:

Number	Term	Document
1	Ram	<2,(1:1),(3:5)>
2	And	<2,(1:2),(2:2)>
3	Shayam	<2,(1:3),(4:1)>
4	Are	<2,(1:4),(2:4)>
5	Friend	<1,(1:5)>
6	Priya	<2,(2:1),(3:1)>
7	Rita	<2,(2:3),(4:4)>
8	Enemy	<1,(2:5)>
9	Is	<2,(3:2),(4:2),>
10	Mother	<1,(3:3)>
11	Father	<1,(4:3)>
12	Of	<2,(3:4),(4:4)>

Table 2.7 Inverted Index with more information

However a more complex inverted file as shown in Fig lI could consist of a list of entries with their weights and pointers to where inside cach document the 'term' occurs

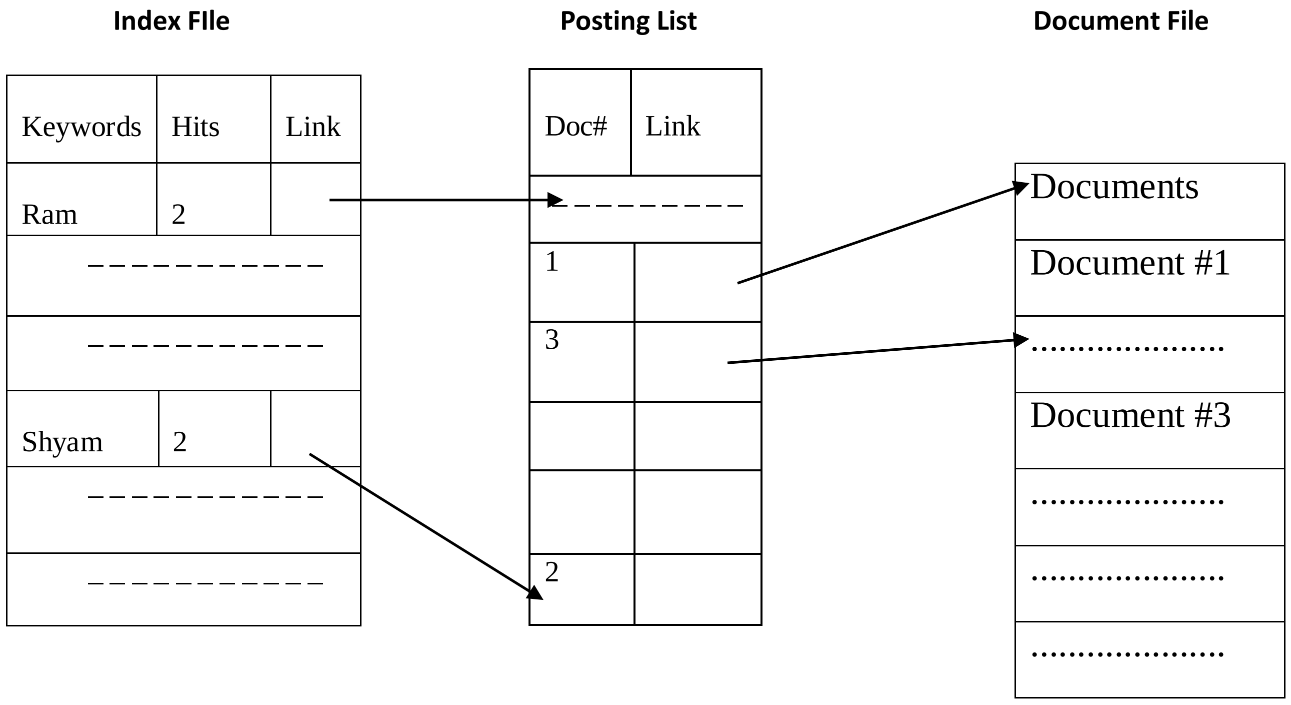


Fig 2.2 Complex Inverted File

Inverted Indices Using Hashing

Hashing: basic plan

Save terms in a key-indexed table (index is a function of the key).

Hash function. Method for computing table index from key.

Issues.

1. Computing the hash function

2. Collision resolution: Algorithm and data structure to handle two keys that hash to the same index.

3. Equality test: Method for checking whether two keys are equal.

Classic space-time tradeoff.

- No space limitation: trival hash function with key as address.
- No time limitation: trivial collision resolution with sequential search.

The procedure goes as follows whenever the search term is given its hash term is calculated using the hash function then this calculated hash value is retrieved from the hash table. The whole process has a hash mapping table which maps each hash value tem to the documents that contains it. An example of a hash function is shown below:

*hashcode() returns a 32-bit int (between -2147483648 and 2147483647)

String s="call",

int code=s.hashCode();

int hash =code%M;.(considering M-8191 for example below)

Document	Text
1	Ram and Shayam are friends
2	Priya and rita are enemy
3	Priya is mother of ram
4	Shayam is father of rita

Table 2.8 Document list

Term	Hash
Ram	7121
And	7188
Shayam	7246
Are	7295
Friend	7394
Priya	7412
Rita	7499
Enemy	7516
Is	7592
Mother	7645
Father	7699
Of	7745

Table 2.9 Mapping of terms to their hash value

Hash	Document
7121	<2,(1:1),(3:5)>
7188	<2,(1:2),(2:2)>
7246	<2,(1:3),(4:1)>
7295	<2,(1:4),(2:4)>
7394	<1,(1:5)>
7412	<2,(2:1),(3:1)>
7499	<2,(2:3),(4:4)>
7516	<1,(2:5)>
7592	<2,(3:2),(4:2),>
7645	<1,(3:3)>
7699	<1,(4:3)>
7745	<2,(3:4),(4:4)>

Table 2.10 Mapping of terms to their hash value with more information

Inverted File Using Sorted array

The production of sorted array inverted files can be divided into two or three sequential steps

First, the input text must be parsed into a list of words along with their location in the text This is usually the most time consuming and storage consuming operation in indexing

Second, this list must then be inverted, from a list of terms in location order to a list of terms ordered for use in searching (sorted into alphabetical order, with a list of all locations attached to each term).

(optional) third step is the postprocessing of these inverted files, such as for adding term weights, or for reorganizing or compressing the files.

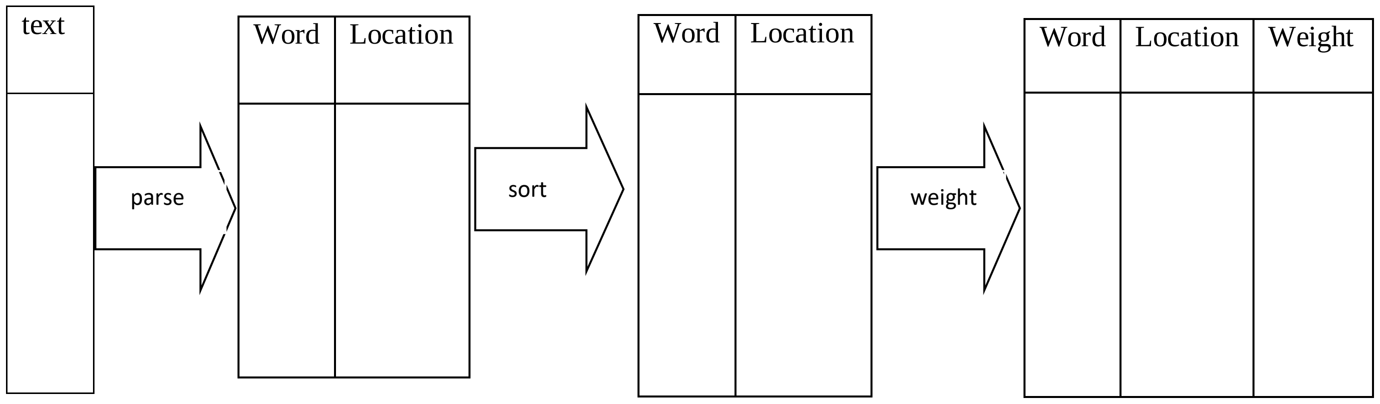


Figure 2.3 Overall schematic of sorted array inverted file creation

Creating the initial word list requires several different operations,

The word list resulting from the parsing operation (typically stored as a disk file) is then inverted. This is usually done by sorting on the word (or stem), with duplicates retained. Even with the use of high-speed sorting utilities, however, this sort can be time consuming for large data sets. One way to handle this problem is to break the data sets into smaller pieces, process each piece, and then correctly merge the results. Note that although only record numbers are shown as locations. typically inverted files store word location

Document	Text
1	Ram and Shayam are friends
2	Priya and rita are enemy
3	Priya is mother of ram
4	Shayam is father of rita

Table 2.11 Document

Term	Doc. No.
Ram	1
And	1
Shayam	1
Are	1
Friend	1
Priya	2
And	2
Rita	2
Are	2

Enemy	2
Priya	3
Is	3
Mother	3
Of	3
Ram	3
Shayam	4
Is	4
Father	4
Of	4
Rita	4

Table 2.12 List

Term	Doc. No.
And	1
And	2
Are	1
Are	2
Enemy	2
Father	4
Friends	1
Is	3
Is	4
Mother	3
Of	3
Of	4
Priya	2
Priya	3
Ram	1
Ram	3
Rita	2
Rita	4
Shayam	1
shayam	4

Table 2.13 Sorted list

Term	No. of posting	Posting file
And	1	<1,1,><2,1>
Are	1	<1,1,><2,1>
Enemy	2	<2,1>
Father	4	<4,1>
Friends	1	<1,1>
Is	3	<3,1><4,1>
Mother	3	<3,1>
Of	3	<3,1><4,1>
Priya	2	<2,1><3,1>
Ram	1	<1,1><3,1>
Rita	2	<2,1><4,1>
shayam	1	<1,1><4,1>

Table 2.14 Dictionary and postings file from the last example

A common search technique is to use a binary search routine on the file to locate the query words. This implies that the file to be searched should be as short as possible, and for this reason the single file shown containing the terms, locations, and (possibly) frequencies is usually split into two pieces[33). The first piece is the dictionary containing the term. statistics about that term such as number of postings, and a pointer to the location of the postings file for that term. The second piece is the postings file itself, which contains the record numbers (plus other necessary location information) and the (optional) weights for all occurrences of the term. The main disadvantage of this approach is that updating the index for appending a new keyword is expensive[34).

B-Tree inverted Index Construction

The leaf nodes of the inverted index are stored in an order such that the logical position does not correspond to the actual position on the disk.

In B-trees internal (non-leaf) nodes contain a number of keys, greater than m and smaller than 2m whrere m is given tree depth[30]. A node gives birth to I+no, of keys new node Each key value K is associated with two pointers one points directly to the block that contains the entry corresponding to ki (denoted (KI), while the second one points to subtree with keys greater than Ki and less than 1 Scarching for a key K in a B-tree is recursive procedure starts at the B-tree root node IFK is found, the search stops. Otherwise IFK is smaller than the leftmost key in the node, the search proceeds followingthe node's leftmost pointer if K is greater than the right most key in the node, the search proceeds following the rightmost pointer if K is comprised between two keys of the node, the search proceeds within the corresponding node

The maintenance of B-trees requires two operations: insertion and deletion When the insertion of a new key value cannot be done locally to a node because it is full (ie, it has reached the maximum number of keys supported by the Btree structure), the median key of the node is identified, two child nodes are created, containing the same number of keys, and the median key remains in the current node[35). When a key is deleted, Two "nearby" nodes have entries that could be condensed into a single node in order to maintain a high node filling rate and minimal paths from the root to the leaves this is the merge procedure it causes a decrease of pointers in the upper node, one merge may recursively cause another merge. A B-tree is kept balanced by requiring that all leaf nodes be at the same depth.

This depth will increase slowly as elements are added to the tree, but an increase in the overall depth is infrequent, and results in all leaf nodes being one more node further away from the root.

The index file can be created using btree or b+tree simply and is very efficient in secondary storage retrieval of data but how a B+-tree may store a search-key value multiple times, up to the height of the tree, a B-tree index seeks to eliminate this redundancy.

In a B-tree nonleaf node, in addition to the child node pointer, we also store a pointer to therecords or bucket for that specific value (since it doesn't repeat)

This results in a new constant m. m <n, indicating how many search-key values may reside in a nonleaf node.

Updates are also a bit more complex: ultimately, the space savings are marginal, so B+ wins on simplicity

Document	**Text**
1	Ram and Shayam are friends
2	Priya and rita are enemy
3	Priya is mother of ram
4	Shayam is father of rita

Table 2.15 Document

Term	**Document**
Ram	<2,(1:1),(3:5)>
And	<2,(1:2),(2:2)>
Shayam	<2,(1:3),(4:1)>
Are	<2,(1:4),(2:4)>
Friend	<1,(1:5)>
Priya	<2,(2:1),(3:1)>
Rita	<2,(2:3),(4:4)>
Enemy	<1,(2:5)>
Is	<2,(3:2),(4:2),>
Mother	<1,(3:3)>
Father	<1,(4:3)>
Of	<2,(3:4),(4:4)>

Table 2.16 Inverted Index

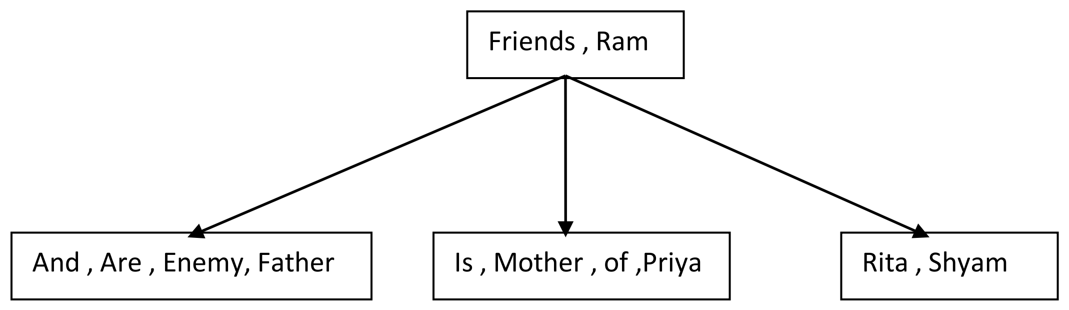


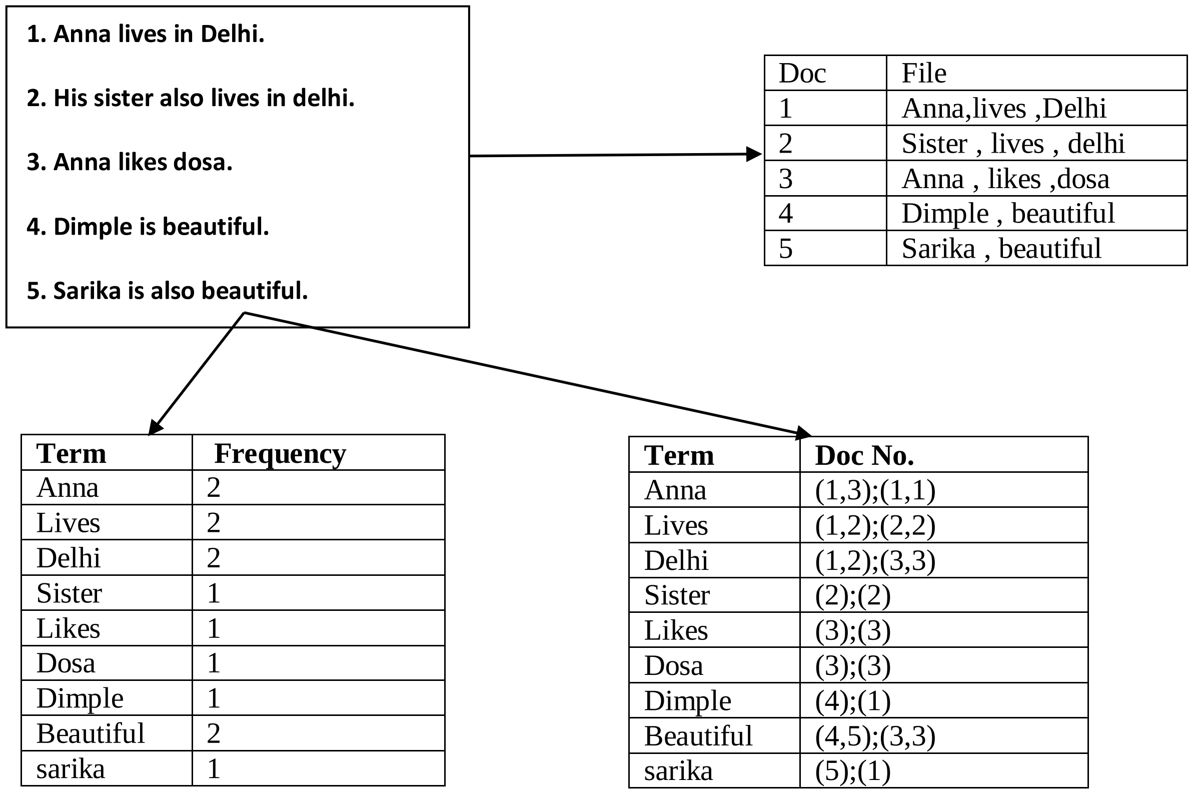
Figure 2.4 B-Tree

2.4.3 DATA STRUCTURE FOR INDEXING

Some important text based indexing are described here. These indexing are suffix arrays, signature files and inverted index, full text indexing, vector space model, latent semantic indexing[29]. Among these inverted index is most popular and important index in IR. So inverted index is used widely in IR and it is described in detail followed by description of other techniques

Inverted file structure

Inverted file index is the popular method. It associates web documents with key words. To find out web documents, searching process findsterms first then retrieve document list corresponding to the terms from inverted index. It contains mainly 3 components



1. Anna lives in Delhi.

2. His sister also lives in delhi.

3. Anna likes dosa.

4. Dimple is beautiful.

5. Sarika is also beautiful.

Doc	File
1	Anna,lives ,Delhi
2	Sister , lives , delhi
3	Anna , likes ,dosa
4	Dimple , beautiful
5	Sarika , beautiful

Term	**Frequency**
Anna	2
Lives	2
Delhi	2
Sister	1
Likes	1
Dosa	1
Dimple	1
Beautiful	2
sarika	1

Term	**Doc No.**
Anna	(1,3);(1,1)
Lives	(1,2);(2,2)
Delhi	(1,2);(3,3)
Sister	(2);(2)
Likes	(3);(3)
Dosa	(3);(3)
Dimple	(4);(1)
Beautiful	(4,5);(3,3)
sarika	(5);(1)

Figure 2.5 An inverted index

Documents, dictionary and inverted list as shown in the figure 2.5.

- First step is to create the document file containing words which are extracted from web documents and a unique number is assigned to each web document.
- Second step is to extract the term and forms the dictionary. Stemming of keywords occur. It is useful when misspelled words are there. Search on dictionary can be finished in
  O(l) and there may be some other storage and time saving measurements to organize dictionary terms in much better way according to (Berry and Browne .2005)
- Last step is the formation of inverted list in which each unique searching term has an associated posting list and it can take more space in proportion to the number of documents.

Whole searching process will be in 3 steps

- Extract searchable term from dictionary
- Extract posting list i.e. all the documents containing this word from inversion list
- Intersection operation on occurrence table to produce final result

Signature files

It is another indexing technique. It uses filter refinement approach for searching of textual word in a document. This is quick and dirty approach because it depends on data structure Some false match appear in the results of query then check can be applied for removing the false match. In this technique fixed length code as in figure 2.15 is designed for each record Record is a test document or it's a part of a document. During searching process it is searched in the records for fixed length code, which is a signature Records which qualifies the searching are sent for further scanning and left are discarded

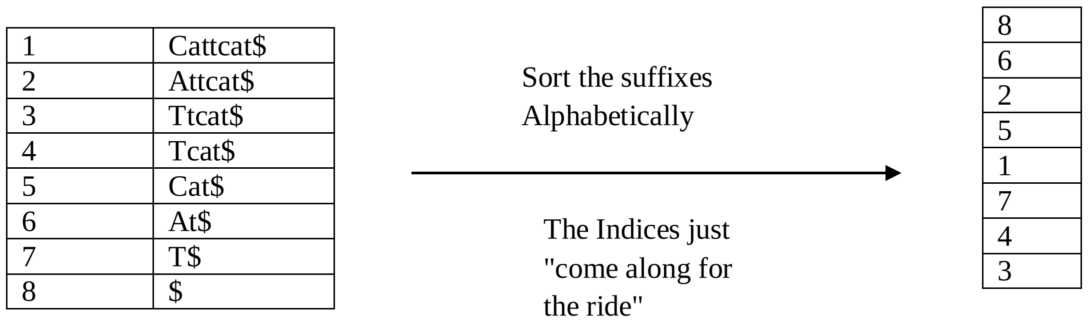
Record	Signature
Model	110 101 001 111
Winner	010 011 110 100
Block Signature	110 111 111 111

Figure 2.6 Fixed length code

It is also useful for conjunction and disjunction queries. 3 methods could be used to improve the performance. According to (Rigaux et al., 2002): compression, vertical partitioning and horizontal partitioning. It is faster and require less space than inverted files and also friendlier for insertion and deletion of records. Only new signature has to be added with the addition of new records without updation of whole index but these are slow for large database. Thus processing of textual query using signature file is proportional to number of items and it is useful only for medium size database, low query frequency and application under distributed environment. This is the reason it is not used so frequently and inferior to inverted files in terms of indexing speed and functionality

Suffix arrays

It is used for searching of large body of text as in figure 2.21, 222, 223. It is an array of integers indicating the starting position of all suffixes of a string in alphabetical order. To search any string binary search can be applied using suffix array(31]. Suffix array is an implementation of suffix tree. Suffix array takes more time but more space efficient.

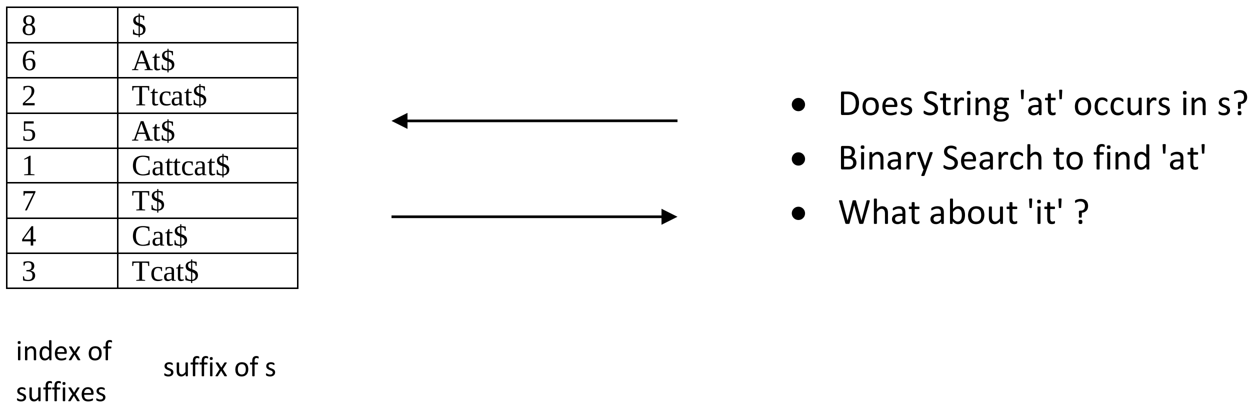


1	Cattcat$
2	Attcat$
3	Ttcat$
4	Tcat$
5	Cat$
6	At$
7	T$
8	$

Sort the suffixes Alphabetically
The Indices just "come along for the ride"

8
6
2
5
1
7
4
3

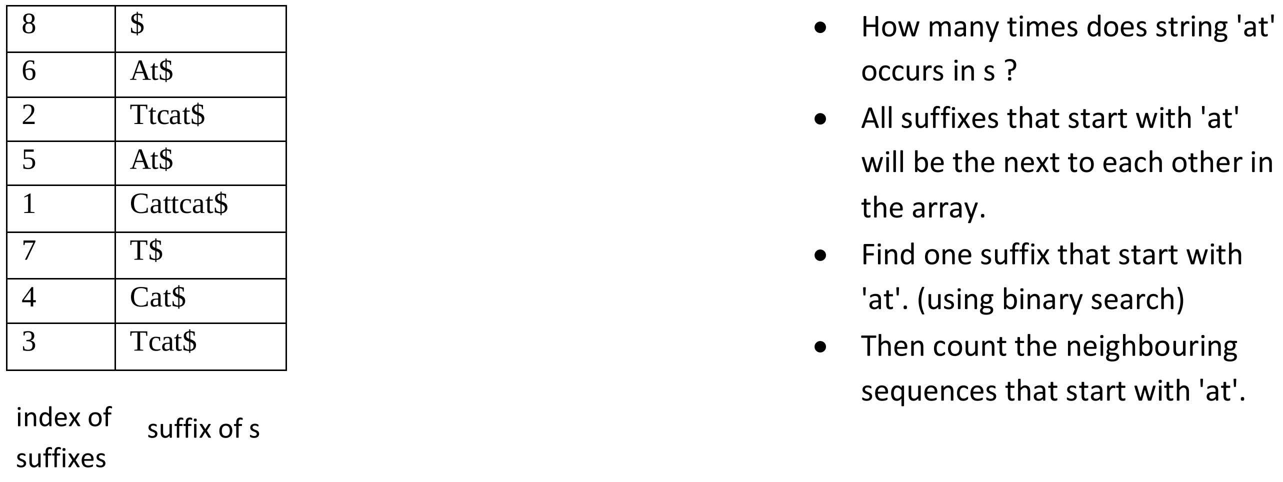
Figure 2.7. Sorting of suffix alphabetically



8	$
6	At$
2	Ttcat$
5	At$
1	Cattcat$
7	T$
4	Cat$
3	Tcat$

- Does String 'at' occurs in s?
- Binary Search to find 'at'
- What about 'it' ?

Figure 2.8. Binary search



8	$
6	At$
2	Ttcat$
5	At$
1	Cattcat$
7	T$
4	Cat$
3	Tcat$

- How many times does string 'at' occurs in s ?
- All suffixes that start with 'at' will be the next to each other in the array.
- Find one suffix that start with 'at'. (using binary search)
- Then count the neighbouring sequences that start with 'at'.

Figure 2.9 Searching of Sequences start with at

But it is good in some cases like searching for phrases searching by regular expression, longest repetition and string searching soon.

2.4.4 WAVELET TREE

Wavelet tree is multipurpose data structure that can be used many directions of research for different purposes. Applications of wavelet tree and its uses are explained in (Wavelet trees for all and Wavelet trees Survey). In explain the characteristics of wavelet tree data structure in different directions and useful results in terms of space-time complexity32). Wavelet tree was invented to reduce space required to store sequences and alphabets [Roberto G. Gupta A, and Vitter JS 2003) proposed Wavelet tree for compressed suffix array and full text indexing with minimum space and search time[33].

Stated that a wavelet tree can be treated as a representation of a sequence, representation of reordering of elements and representation of a grid of points(34),

(Navarro G, Puglisi SJ, Valenzuela D 2011) proposed algorithm for document retrieval but they not explained about implementation of document retrieval. But as per best of my knowledge inverted index was not defined and implemented using wavelet tree.

It explains the applications of wavelet tree in different areas such as strings, set of rectangles inverted indexes, document retrieval, information retrieval and representation of grids[35]

Proposed wavelet matrix for alternative representation of large alphabets. It is comparatively faster but, bit complex to construct.

Working of Wavelet Tree: Working of wavelet tree can be classified in the following ways

Data Structure:

Let w[1,x]=xlx2...xn be a sequence of symbol xi ∑; where ∑ is the finite alphabet of size ki.e∑=[1..x].

It Support Access/Rank Select on sequences on a finite alphabet

Reduces to operations on bit vectors by recursively partitioning the alphabet

Maintaining an index sequence of alphabets for retrieval in compressed space.

Succinct representation of strings

Having string S ,|S|=n , S∈∑* , number of bits=σ

required: nlogσ + O(nlogσ)

Supports queries in O(logσ)

There are three main operations of Wavelet tree

1. rankc(S, i) number of occurrences of symbol c in S[1..i-1]

2. selectc(S. i) position in S of the ith occurrence of c.

3. S[i]-return character at position i

Basic Steps of Constructing Wavelet trees

Constructing a Wavelet Tree: The tree is defined recursively as follows: which includes as following steps.

1. Take the alphabet of the string, and encode the first half as 0, the second half as 1: I,e {a,b,c,d) would become (0,0,1,1);

2. Group each 0-encoded symbol, (a,b), as a sub-tree

3. Group each l-encoded symbol, fcd), as a sub-tree

4. Reapply this to each sub tree recursively until there is only one or two symbols left(when a 0 or 1 can only mean one thing)

Example:

For the string K=”majatamanakshay" it has alphabet (a,h,j,k,m,n,s,t).

Partition the string alphabet

Ahjk|mnst if alphabet is not equal put +1 in right

Now String

For left character assign 0 and for right leg

Ahjk |mnst

0000|1 111

Example of Wavelet tree:

Keyword	**Document**
We	1
Love	1,3
India	1,2
Is	2
Our	2
Home	2,4
I	3,4,5
My	3
Something	4
Don't	4
Know	4,5
Want	5
To	5

Table 2.17 Inverted File

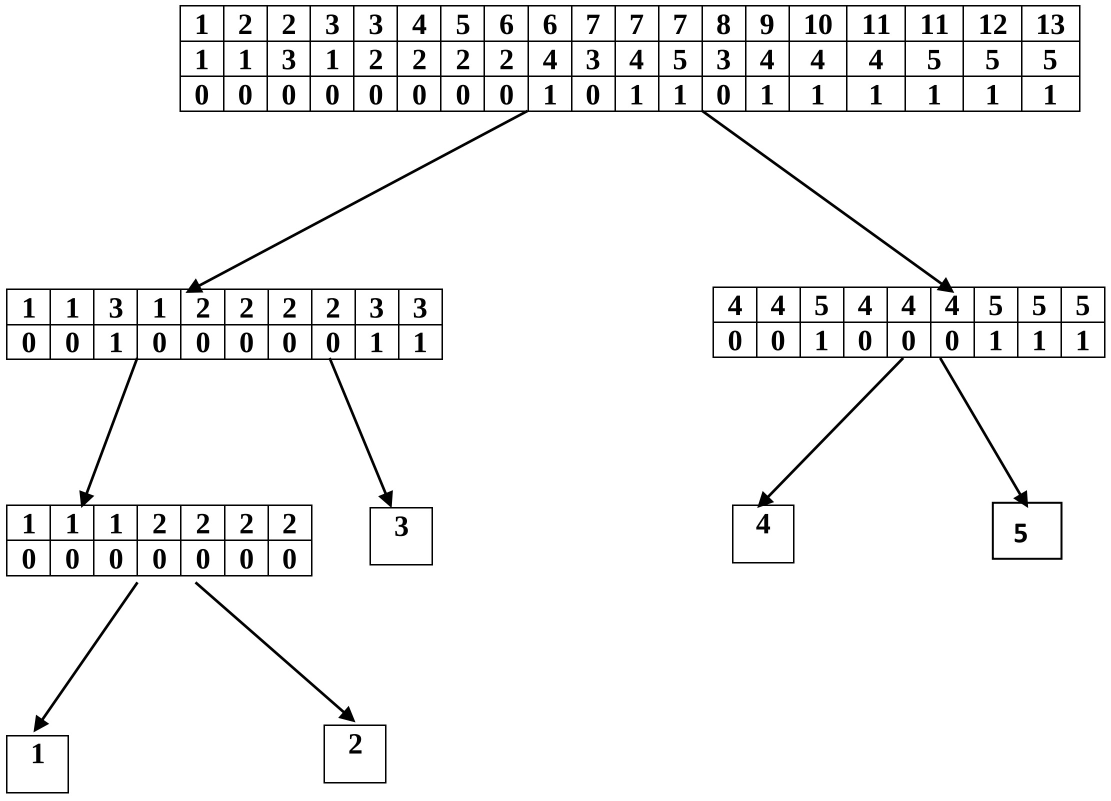


Figure 2.10 Wavelet Tree

Consider the documents presented in Table-1. The wavelet tree for it is shown in Figure 5. Let we want to find keyword "home", whose keyword id is 6. We start with root node of wavelet tree and check the associated binary code with keyword id 6. The keyword id 6 occurs 2 times that means "home" keyword is present in 2 documents

For Ist keyword: binary code is 0 (that means we have to move left) rank of 0-> 8. Move to left child and find 8th element ie 0. rank of 0-> 7 since its 0 move to left child and find 7th element i.e. 1, rank of -> 4 since its I move to right, now leaf node reached and document id is "2". For 2nd keyword: binary code- (that means we need to move right) rank of I-> 1. Move to right child and find 1st element i.e. 0. rank of 0-> since its 0 move to len child. now leaf node reached and document id is "4". Hence we find the document id 2 and 4 for keyword "Home"

CHAPTER 3

RELATED WORK

3.1 INTRODUCTION

A web search engine is a software system that is designed to search for information on the World Wide Web. The search results are generally presented in a line often referred to as search engine results pages (SERPS). The infomation may be a mix of web pages, images, and other types of files. Some search engines also mine data available in databases or open directories. Unlike web directories, which are maintained only by human editors, search engines also maintain real time information by running an algorithm on a web crawler.

The most important phase in working of search engine are **Searching, Indexing** and **Ranking** operations. Search engines are extremely useful information retrieval tools. Information retrieval is a general term that is used to identify all those activities that enable us to choose from a given collection of documents. Information retrieval today is done on the basis of numerous textual and geographic queries having both the textual and spatial components. The textual queries of any IRS can be resolved using the indexes and an inversion list.

Different kinds of data structures can be used to implement the index. Sorted arrays, B-tees, Hash tables, R-Trees, R+ trees, R* Trees and Wavelet tress can be used to create the index. Various issues such as the time required to create the index, the space occupied by an inverted file, the time required for retrieval arise when talking about efficient data structures for the indexing algorithm.

3.2 SEARCHING

Web search engines have their ancestors in the information retrieval (IR) systems developed during the last many decades. In many ways, search engines have become the most Important tool for our Information seeking. Due to their tremendous economic value, search engine companies constantly put major efforts to improve their search results.

In June 1993, Matthew Gray, then at MIT, produced what was probably the

first Web Robot, the Perl —based World Wide Web Wanderer, and used it to generate an index called 'Wandex‘. The purpose of the Wanderer was to measure the size of the World Wide Web, which it did until late 1995 The web's second search engine Aliweb appeared in November 1993.

One of the first "all text" crawler-based search engines was WebCrawler, whichcame out in 1994. Unlike its predecessors, it allowed users to searchfor any word in any webpage, which has become the standard for all major search engines since. It was also the first one widely known by the public. Also 1n 1994, Lycos (which started at Carnegie Mellon University) was launched and became a major commercial endeavor.

In 1996, Netscape was looking to give a single search engine an exclusive deal as the featured search engine on Netscape's web browser. There was so much interest that instead Netscape struck deals with five of the major search engines: for $5 million a year, each search engine would be in rotation on the Netscape search engine page. The five engines were Yahool, Magellan, Lycos, Infoseek, and Excite.

Around 2000, Google's Search Engine rose to prominence. The company achieved better results for many searches with an innovation called PageRank, as was explained in the paper Anatomy of a Search Engine written by W and M the later founders of Google. This iterative algorithm ranks web pages based on the number and PageRank.

3.3 INDEXING

Indexing is a way to optimize performance of a database by minimizing the number of disk accesses required when a query is processed. An index or database index is a data structure which is used to quickly locate and access the data in a database table. These are created using some database columns. It refers to various methods for indexing the contents of a website or of the Internet as a whole. Individual websites or intranets may use a back-of-the-book index, while search engines usually use keywords and metadata to provide a more useful vocabulary for Internet or onsite searching. With the increase in the number of periodicals that have articles online, web indexing is also becoming important for periodical websites.

Back-of-the-book-style web indexes may be called "web site A-Z indexes". The implication with "A-Z" is that there is an alphabetical browse view or interface. This interface differs from that of a browse through layers of hierarchical categories (also known as a taxonomy) which are not necessarily alphabetical, but are also found on some web sites. Although an A-Z index could be used to index multiple sites, rather than the multiple pages of a single site, this is unusual. Metadata web indexing involves assigning keywords or phrases to web pages or web sites within a metadata tag (or "meta-tag") field, so that the web page or web site can be retrieved with a search engine that is customized to search the keywords field. This may or may not involve using keywords restricted to a controlled vocabulary list. This method is commonly used by search engine indexing.

3.4 RANKING

Ranking is performed using PageRank, It is a link analysis algorithm and it assigns a numerical weighting to each element of a hyperlinked set of documents, such as the World Wide Web, with the purpose of "measuring" its relative importance within the set. The algorithm may be applied to any collection of entities with reciprocal quotations and references. The numerical weight that it assigns to any given element *E* is referred to as the *PageRank of E* and denoted by Other factors like *Author Rank* can contribute to the importance of an entity.

A PageRank results from a mathematical algorithm based on the webgraph, created by all World Wide Web pages as nodes and hyperlinksas edges, taking into consideration authority hubs such as cnn.com or usa.gov. The rank value indicates an importance of a particular page. A hyperlink to a page counts as a vote of support. The PageRank of a page is defined recursively and depends on the number and PageRank metric of all pages that link to it ("incoming links"). A page that is linked to by many pages with high PageRank receives a high rank itself.

Numerous academic papers concerning PageRank have been published since Page and Brin's original paper. In practice, the PageRank concept may be vulnerable to manipulation. Research has been conducted into identifying falsely influenced PageRank rankings. The goal is to find an effective means of ignoring links from documents with falsely influenced PageRank.

CHAPTER 4

PROPOSED WORK

4.1 PROBLEM DEFINITION

These days search engines are very popular as we can access the large amount of information available on the World Wide Web. So there is a need of efficient text indexing method. Search engines are used for document and geographical location retrieval, they use inverted indexes for this. In the past the Signature files and the Inverted files were used as indexing techniques. In comparison to inverted files, Signature files require larger space and are expensive in construction. Use of sorted array although guarantees faster access but less efficient updation when compared to B-tree. So, these indexes can be created using different data structures like B tree (for textual indexing), B+ tree (for textual indexing) and wavelet tress. Moreover wavelet trees are efficient data structures as they take less amount of memory storage for storing the inverted index, but the time taken for creation of inverted index is more in wavelet trees as compared to other data structures like B/B+ trees.

4.2 OBJECTIVE

The main objective of our project is to build a inverted index by using parallel wavelet tree , which takes less amount of space to store the inverted index and by using parallelism the speed also enhanced as compared to the serial one.

This project is composed of web based textual search system which takes queries from users and show to users the url of the documents.

In our project we have used efficient indexing techniques based solely on the parallel wavelet tree structure for textual indexing although other data structure structure are also shown for comparing the efficiency.

4.3 WAVELET TREE BASED INDEXING

4.3.1 WAVELET TREE

Wavelet tree(Navarro, G. (2012)-[13]) is a data structure used to stores the data in form of bit representation, this way of storage leads to lesser space utilization. On comparison with data structures like B-tree and R-tree, wavelet tree takes much less space for storage with similar dataset.
Moreover, wavelet tree can store both type of data i.e. both textual and spatial.
For any document there is an inverted file which consist a list of keywords along with the posting which consist occurrence positions of the respective keywords. The wavelet tree can be constructed based on the relative positions of terms in the document, the nodes of the tree represents the keywords or terms positions. Each position assigned with a range values from 0 or 1. Then the tree is divided into two parts known as the child of tree. The left child consist all the 0 indexed values and the right child consist the 1 indexed values. Then again the positions assigned the values 0 or 1. 0 is assigned to the lower numbered positions and 1 is assigned to the higher numbered positions.

4.3.2 INVERTED INDEXING

An inverted index contains a postings lists, it contains the list of unique words that comes in the collection of different words. A postings list has information about that unique term which includes the document id and number of occurrences of the term in that particular document(Inverted indexes: Types and techniques, Ajit Kumar Mahapatra , Sitanath Biswas July 2011)18]. Payloads can be more complex if we attach the position where the words have occurred. Lets take an example given in fig.[4.1], for better understanding,

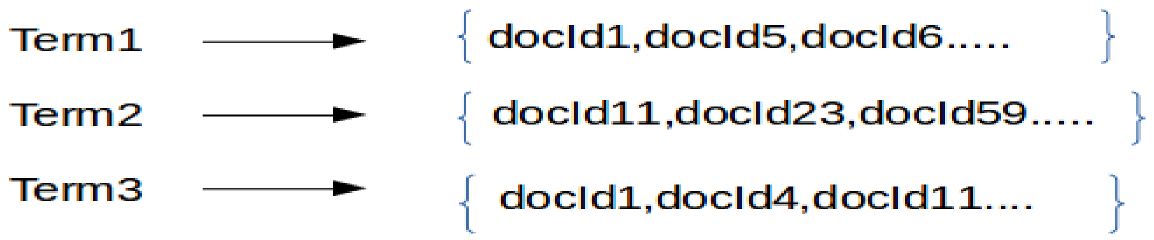

Term1
{ docId1,docId5,docId6..... }
Term2
{ docId11,docId23,docId59..... }
Term3
{ docId1,docId4,docId11.... }

Fig. 4.1- This figure shows the unique terms with the list of document id's of the documents in which they are present

In general we can identify the document by the document number attached, if there are n documents then these document number range from {1,2,..n}. This concept can be seen from fig.[4.2]

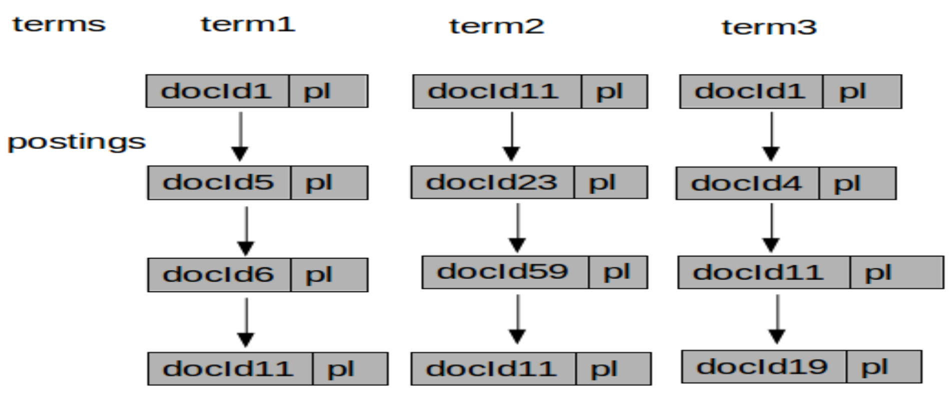


Fig.4.2:- This figure shows how the different terms present in different documents are taken together and represented as list. Each of the list elements contains two parts, document id and the payload information of how many occurrences of that word were there.

4.3.2.1 MAPPER :

The mapper module of inverted index takes in all the unique words from each document and assign them a document id to which those words belong along with their number of occurrences. For this purpose we make the use of pair data structure available in the C++ library. We iterate over all the documents present in the repository, extracting out the unique words and making a pair of that particular word with the document id in which that word is found. The output that we get from mapper is a vector that contains pairs of the word with their corresponding document id. Fig.[4.3] shows how the mapper works

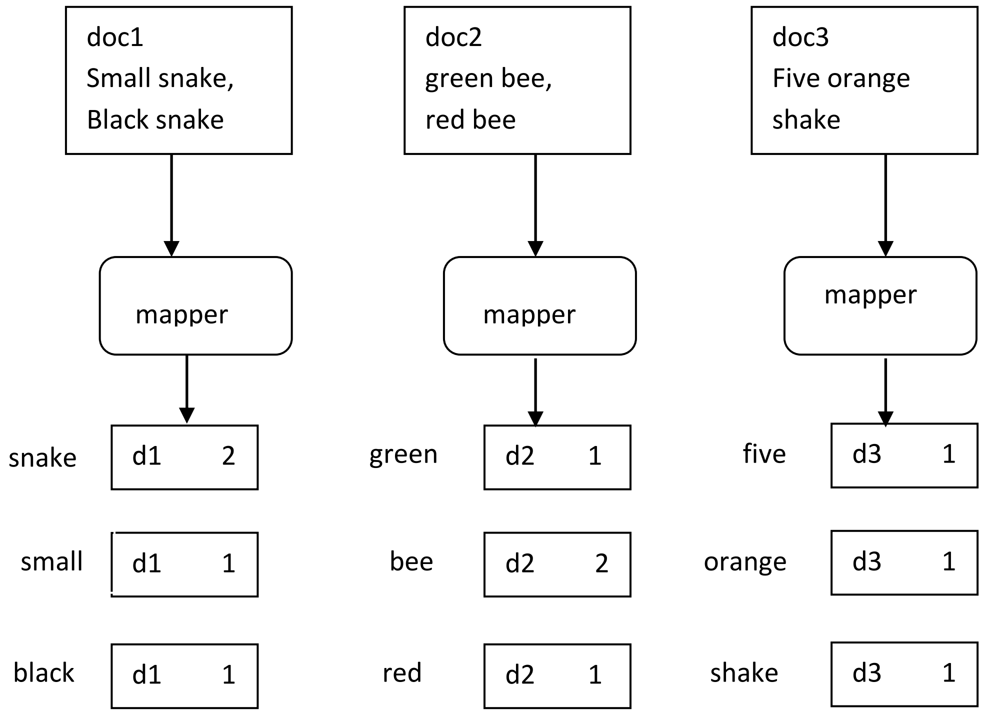


Fig. 4.3-This figure shows the Working of mapper

MAPPER ALGORITHM

MAPPER() :

```
vector<string, id> collection
// global vector of all words with their doc  id.
1.  array[]= {d1,d2,d3,…dn};
 // array of documents.
2. s := array[].size()
3.  string w;     //words in document.
4. fori :=0 to s :
5.        read document[ i ]
6.        while ( $\forall w : w \in$document[i]) :
           //read every document
7.        make_pair p< word, i +1>
          //pair document with its doc id.
8.        collection.push( p )
9.        end while
10. end for
```

Explanation of Mapper() :-

In this algorithm first we create a global vector named collection of type <string,int>.
Then we create an array named array[] which contains documents. Variable 's' contains size of the array[].We create a string named w which used to traverse the words in the documents. Now we iterate a for loop with s number of iterations i.e number of documents. In each iteration each document is read and words of that document listed into the pair of <word,documentId>.And the elements of that pair is added in the global vector described in the beginning.
By this we get a vector which stores the word with their document id.

4.3.2.2 REDUCER :

This module of inverted index takes in the output produced by mapper and combines the document id's of similar words together into a list. We have used wavelet tree in implementing the work of reducer, for this we use pair data structure available in C++ to combine the all words in the document with their document id and create a vector of pairs.In the algorithm, we associate the pair of <word, doc id> with a bit 0 or 1 according to its position in left half or right half in vector. At each level the bit representation changes according to the working of wavelet tree. This can be easily seen from the fig.[5]

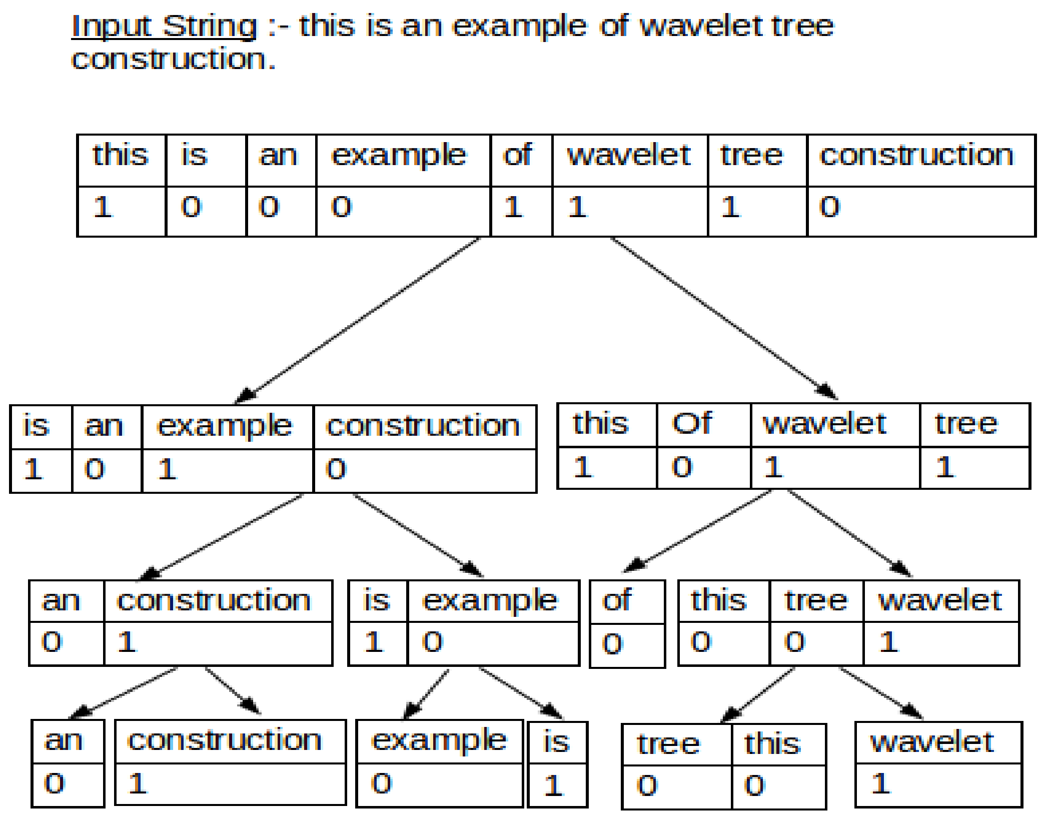
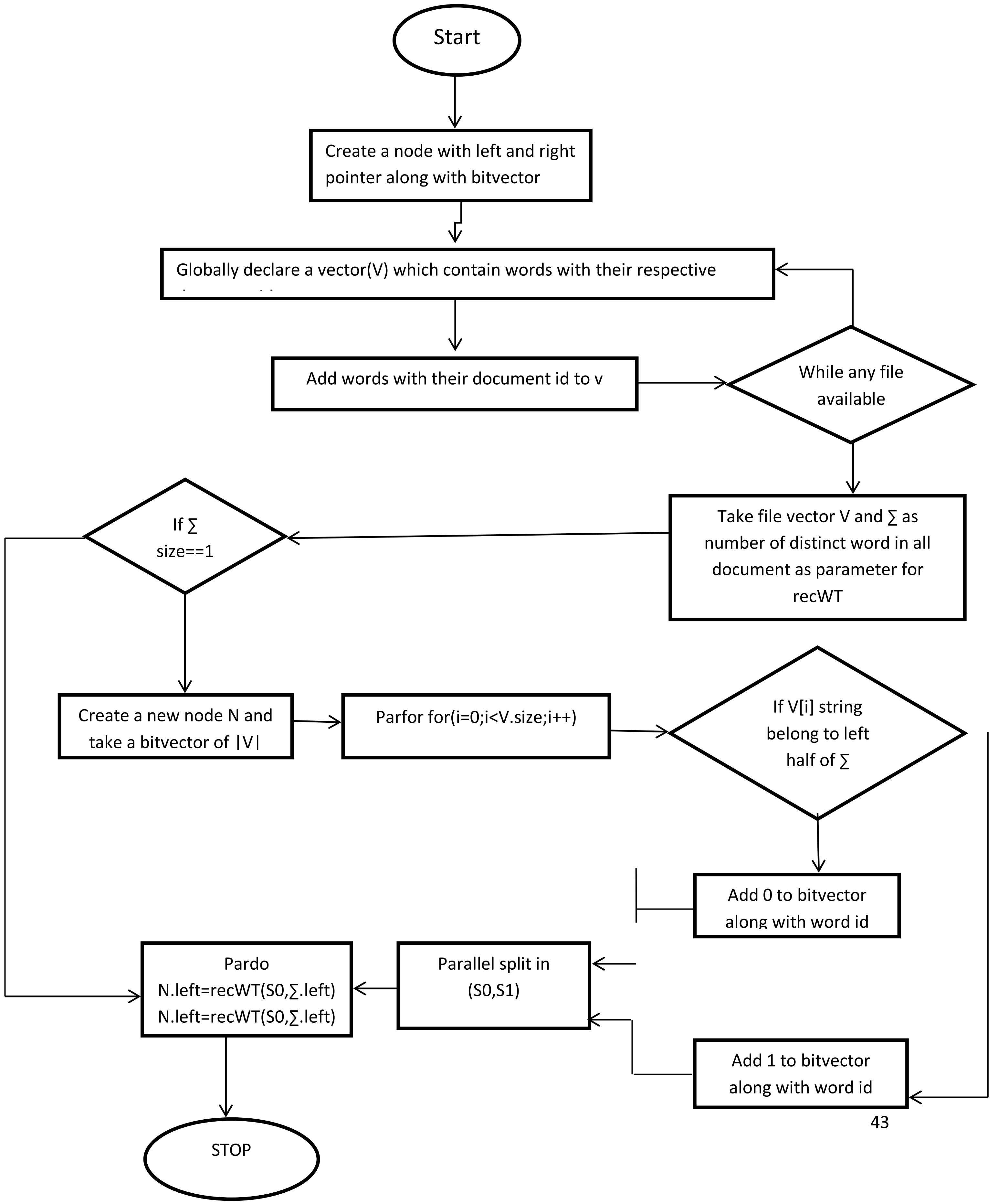


Fig. [4.4]- Construction of wavelet tree for the string that contains different words

FLOWCHART AND ALGORITHM


Start
Create a node with left and right pointer along with bitvector
Globally declare a vector(V) which contain words with their respective
Add words with their document id to v
While any file available
Take file vector V and ∑ as number of distinct word in all document as parameter for recWT
If ∑ size==1
Create a new node N and take a bitvector of |V|
Parfor for(i=0;i<V.size;i++)
If V[i] string belong to left half of ∑
Add 0 to bitvector along with word id
Add 1 to bitvector along with word id
Parallel split in (S0,S1)
Pardo
N.left=recWT(S0,∑.left)
N.left=recWT(S0,∑.left)
STOP

REDUCER ALGORITHM :

```
1.  Create a node with following fields
2.  Left pointer  //pointer to left subtree
3.  Right pointer // pointer to right subtree
4.  BitVector          //A bit vector
5.   node *RwTree(string,Σ = [a,z]) {
6.     if a = z: return leaf labelled with a      // base condition for termination(only single word)
7.     v := root node                              // create a new node
8.    v.bitvector := BitVector of size |S|      // take a bitvector of size |S|
9.     parfor(inti=0;i<|S|-1;i++) {
10.            if(S[i] <= (a+z)/2) {
11.                v.bitvector[i] = 0;                 // belong to the left subtree
12.              }
13.           else {
14.               v.bitvector[i] = 1;                 // belong to the right subtree
15.          }
16.     }
17.     //Split the String (S) into two parts {S0,S1}
18.     {S0,S1} = parallelSplit(S,v.bitvector)
19.      Pardo:
        // Build the left and right subtree in    parallel
20.      v.leftChild = RwTree (S0,[a, (a+z)/2 ])        // make the left subtree recursively
21.     v.rightChild = RwTree (S1,[(a+z)/2 +1,z]) // make the right subtree recursively
22.      return v     // Return the bitvector
23.   }
```

WORKING OF ALGORITHM:-

As described in Flow Chart. Each node of the wavelet tree contains these three fields: -

- One left and one right pointer of node type
- One vector 'v' which store the the values as pair of int,int type

We have some global variables. These are:-

- A vector 's' which stores values as pair of string,int type. This variable stores the word and its document id as pair.
- A string W which is used to read words from each file.
- A vector 'word' which stores data as string type. This vector store all the unique words occur in all the documents. This vector is used to assign each word either value 0 or 1.

Now let's see how our algorithm is working. Suppose we have two text files. Firstly we call mapper function , the working of mapper function is as given below :-

In mapper function all the words from all documents are filled in the vector 's' with their document id's. Here each word is read from each document by using W variable , and this word is entered into the vector s with the document is of the current scanned word.

Now the 'word' vector is sorted and unique elements are filled in this vector.

Then a node named root is declared which act as the root of the wavelet tree.

After this rwTree(vector< pair<string,int>>s,intl,intr,intch) function is called which return the pointer to the root which is stored in the 'root'. Now before going to the working of the rwTree let us first understand about the functions which this function calls:

1. **Bitvector :-**

   This function assigns the 0 or 1 to the words in the current node by looking into the 'word' vector. It returns a vector which stores the data as pair<int,int> which contains 0 or 1 in the first field and document id of the corresponding word in the second field.

2. **Parallelsplit :-**
   This function splits the bit variable into the left and right child of the node and returns a vector of vector which contain data as pair<string,int> . This vector contains the left child at the 0th index and right child at 1st index.

Now the rwTree function callls these function in the given order :

- First it store the result of bitvector() function in the 'v' field of current node.
- Then it calls parallelsplit() funtion which return the left and right sub-tree of the node and this is stored in the variable 'vs' .

Then it parallely calls the rwtree function for left and right tree.

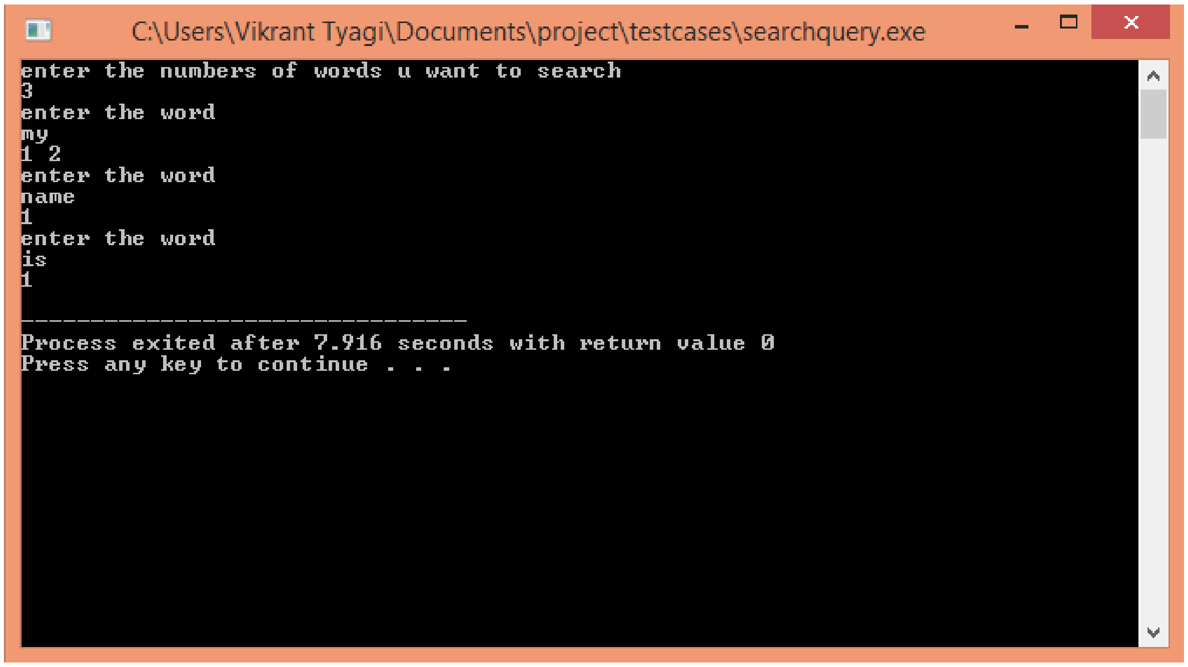
DOCUMENT RETRIEVAL

Here the document id's are retrieved based on the words we are given as an input .

METHOD TO RETRIEVE THE DOCUMENT ID'S

```
void access_query(int idx,int l,int r,node *root)
{
	map<int,int> eliminateDuplicateId;
	int m = (l+r)/2;
	if(root->leaf) {
		vector< pair<int,int>>u = root->v;
		//cout << u.size() << endl;
		for(int i=0;i<u.size();i++) {
			if(eliminateDuplicateId[u[i].second]++ == 0)
				cout << u[i].second << " ";
		}
		return ;
	}
	if(idx >= l && idx <= m) {
		access_query(idx,l,m,root->left);
	}
	else {
		access_query(idx,m+1,r,root->right);
	}
}
```

Here is the screenshot of the working of search method :-


C:\Users\Vikrant Tyagi\Documents\project\testcases\searchquery.exe

```
enter the numbers of words u want to search
3
enter the word
my
1 2
enter the word
name
1
enter the word
is
1

--------------------------------
Process exited after 7.916 seconds with return value 0
Press any key to continue . . .
```

Fig -4.5As shown in the screenshot , word- my is in two files 1 and 2 so the doc id of file as 1 and 2 are printed.

CHAPTER 5

ENVIRONMENTAL SETUP

In this,we have used DevCPP as the development environment and OpenMP libraray to achieve parallelsim in our program.

5.1 DEVCPP

Dev-C++ is a free full-featured integrated development environment (IDE) distributed under the GNU General Public License for programming in C and C++. It is written in Delphi.

It is bundled with, and uses, the MinGW or TDM-GCC 64bit port of the GCC as its compiler. Dev-C++ can also be used in combination with Cygwin or any other GCC-based compiler.

Dev-C++ is generally considered a Windows-only program, but there are attempts to create a Linux version: header files and path delimiters are switchable between platforms.

5.1.1 DEVPACKS

An additional aspect of Dev-C++ is its use of DevPaks: packaged extensions on the programming environment with additional libraries, templates, and utilities. DevPaks often contain, but are not limited to, GUI utilities, including popular toolkits such as GTK+, wxWidgets, and FLTK. Other DevPaks include libraries for more advanced function use. Users of Dev-C++ can download additional libraries, or packages of code that increase the scope and functionality of Dev-C++, such as graphics, compression, animation, sound support and many more. Users can create Devpaks and host them for free on the site. Also, they are not limited to use with Dev-C++ - the site says "A typical devpak will work with any MinGW distribution (with any IDE for MinGW)".

5.1.2 DEVELOPMENTSTATUS

From February 22, 2005 to June 2011 the project was not noticeably active, with no news posted nor any updated versions released. In a 2006 forum post, lead developer Colin Laplace stated that he was busy with real-life issues and did not have time to continue development of Dev-C++.

There are two forks of Dev-C++ since then: wxDev-C++ and the *Orwell* version.

wxDev-C++ is a development team that has taken Dev-C++ and added new features such as support for multiple compilers and a RAD designer for wxWidgets applications.

On June 30, 2011 an unofficial version 4.9.9.3 of Dev-C++ was released by Orwell (Johan Mes), an independent programmer, featuring the more recent GCC 4.5.2 compiler, Windows' SDK resources (Win32 and D3D), numerous bugfixes, and improved stability. On August 27, after five years of officially being in a beta stage, version 5.0 was released. This version also has its own separate SourceForge page since version 5.0.0.5, because the old developer isn't responding to combining requests. On July 2014, Orwell Dev-C++ 5.7.1 was released featuring the more recent GCC 4.8.1 which supports C++11.

5.2 OPENMP

OpenMP (**Open Multi-Processing**) is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most platforms, instruction set architectures and operating systems, including Solaris, AIX, HP-UX, Linux, macOS, and Windows. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior.

OpenMP is managed by the nonprofit technology consortium *OpenMP Architecture Review Board* (or *OpenMP ARB*), jointly defined by a group of major computer hardware and software vendors, including AMD, IBM, Intel, Cray, HP, Fujitsu, Nvidia, NEC, Red Hat, Texas Instruments, Oracle Corporation, and more.

OpenMP uses a portable, scalable model that gives programmers a simple and flexible interface for developing parallel applications for platforms ranging from the standard desktop computer to the supercomputer.

An application built with the hybrid model of parallel programming can run on a computer cluster using both OpenMP and Message Passing Interface (MPI), such that OpenMP is used for parallelism *within* a (multi-core) node while MPI is used for parallelism *between* nodes. There have also been efforts to run OpenMP on software distributed shared memory systems, to translate OpenMP into MPI and to extend OpenMP for non-shared memory systems.

5.2.1 DESIGN

OpenMP is an implementation of multithreading, a method of parallelizing whereby a *master* thread (a series of instructions executed consecutively) *forks* a specified number of *slave* threads and the system divides a task among them. The threads then run concurrently, with the runtime environment allocating threads to different processors.

The section of code that is meant to run in parallel is marked accordingly, with a compiler directive that will cause the threads to form before the section is executed. Each thread has an *id* attached to it which can be obtained using a function (called `omp_get_thread_num()`). The thread id is an integer, and the master thread has an id of *0*. After the execution of the parallelized code, the threads *join* back into the master thread, which continues onward to the end of the program.

By default, each thread executes the parallelized section of code independently. *Work-sharing constructs* can be used to divide a task among the threads so that each thread executes its allocated part of the code. Both task parallelism and data parallelism can be achieved using OpenMP in this way.

The runtime environment allocates threads to processors depending on usage, machine load and other factors. The runtime environment can assign the number of threads based on environment variables, or the code can do so using functions. The OpenMP functions are included in a header file labelled omp.h in C/C++.

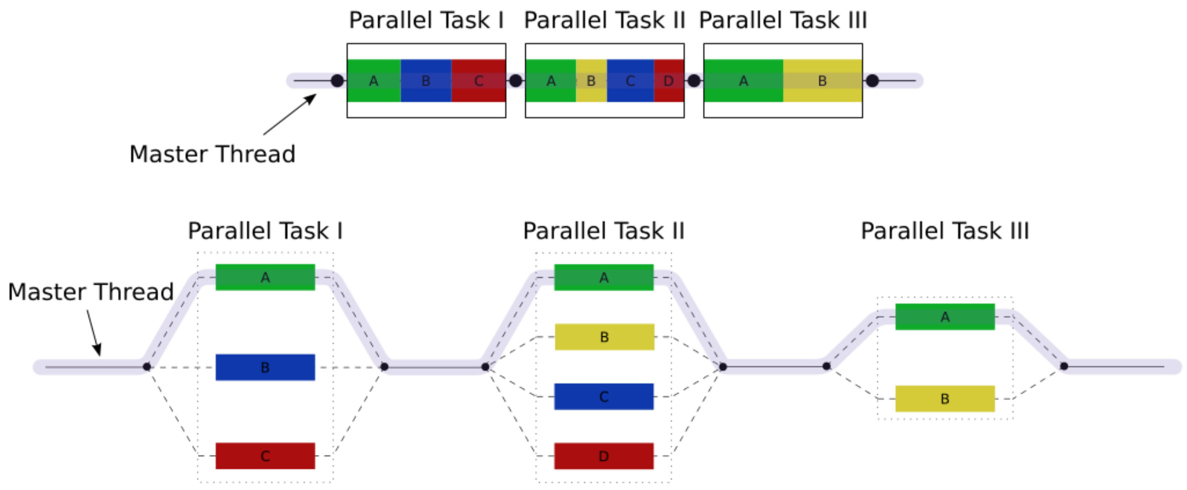


Fig 5.1 An illustration of multithreading

5.2.2 HISTORY

The OpenMP Architecture Review Board (ARB) published its first API specifications, OpenMP for Fortran 1.0, in October 1997. In October the following year they released the C/C++ standard. 2000 saw version 2.0 of the Fortran specifications with version 2.0 of the C/C++ specifications being released in 2002. Version 2.5 is a combined C/C++/Fortran specification that was released in 2005.

Up to version 2.0, OpenMP primarily specified ways to parallelize highly regular loops, as they occur in matrix-oriented numerical programming, where the number of iterations of the loop is known at entry time. This was recognized as a limitation, and various task parallel extensions were added to implementations. In 2005, an effort to standardize task parallelism was formed, which published a proposal in 2007, taking inspiration from task parallelism features in Cilk, X10 and Chapel.

Version 3.0 was released in May 2008. Included in the new features in 3.0 is the concept of *tasks* and the *task* construct,[11] significantly broadening the scope of OpenMP beyond the parallel loop constructs that made up most of OpenMP 2.0.

Version 4.0 of the specification was released in July 2013. It adds or improves the following features: support for accelerators; atomics; error handling; thread affinity; tasking extensions; user defined reduction; SIMD support; Fortran 2003 support.

The current version is 4.5.

Note that not all compilers (and OSes) support the full set of features for the latest version/s.

5.2.3 CORE ELEMENTS

The core elements of OpenMP are the constructs for thread creation, workload distribution (work sharing), data-environment management, thread synchronization, user-level runtime routines and environment variables.

In C/C++, OpenMP uses #pragmas. The OpenMP specific pragmas are listed below.

Thread creation

The pragma *omp parallel* is used to fork additional threads to carry out the work enclosed in the construct in parallel. The original thread will be denoted as *master thread* with thread ID 0.

Example (C program): Display "Hello, world." using multiple threads.

```
#include<stdio.h>
#include<omp.h>

intmain(void)
{
#pragma omp parallel
    printf("Hello, world.\n");
return0;
}
```

Use flag -fopenmp to compile using GCC:

```
$ gcc -fopenmp hello.c -o hello
```

Output on a computer with two cores, and thus two threads:

```
Hello, world.
Hello, world.
```

However, the output may also be garbled because of the race condition (in the case of using C++ std::cout, for example, the example is always true. printf can be or not thread-safe) caused from the two threads sharing the standard output.

```
Hello, wHello, woorld.
rld.
```

Work-sharing constructs

Used to specify how to assign independent work to one or all of the threads.

- *omp for* or *omp do*: used to split up loop iterations among the threads, also called loop constructs.
- *sections*: assigning consecutive but independent code blocks to different threads
- *single*: specifying a code block that is executed by only one thread, a barrier is implied in the end
- *master*: similar to single, but the code block will be executed by the master thread only and no barrier implied in the end.

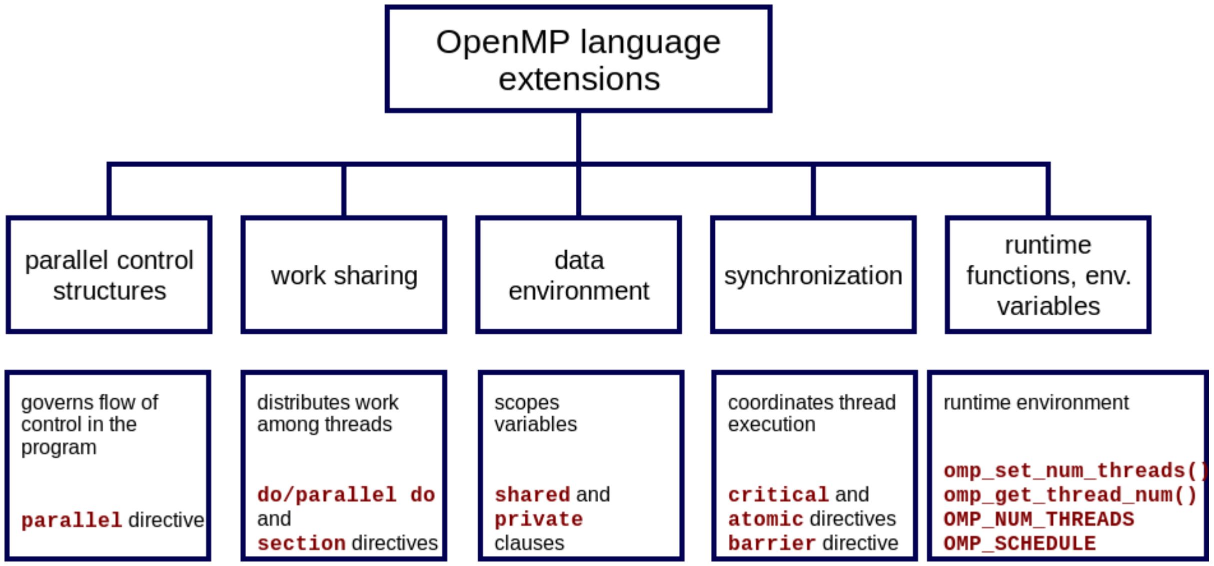
Example: initialize the value of a large array in parallel, using each thread to do part of the work

```
intmain(int argc, char**argv)
{
int a[100000];

#pragma omp parallel for
for (int i =0; i <100000; i++) {
        a[i] =2* i;
    }

return0;
}
```

This example is embarrassingly parallel, and depends only on the value of i. The OpenMP parallel for flag tells the OpenMP system to split this task among its working threads. The threads will each receive a unique and private version of the variable. For instance, with two worker threads, one thread might be handed a version of i that runs from 0 to 49999 while the second gets a version running from 50000 to 99999.



OpenMP language extensions
parallel control structures
work sharing
data environment
synchronization
runtime functions, env. variables
governs flow of control in the program
parallel directive
distributes work among threads
do/parallel do and section directives
scopes variables
shared and private clauses
coordinates thread execution
critical and atomic directives
barrier directive
runtime environment
omp_set_num_threads()
omp_get_thread_num()
OMP_NUM_THREADS
OMP_SCHEDULE

Fig. 5.2 OpenMP Constructs

5.2.4 PERFORMANCE EXPECTATION

One might expect to get an *N* times speedup when running a program parallelized using OpenMP on a *N* processor platform. However, this seldom occurs for these reasons:

- When a dependency exists, a process must wait until the data it depends on is computed.
- When multiple processes share a non-parallel proof resource (like a file to write in), their requests are executed sequentially. Therefore, each thread must wait until the other thread releases the resource.
- A large part of the program may not be parallelized by OpenMP, which means that the theoretical upper limit of speedup is limited according to Amdahl's law.
- N processors in a symmetric multiprocessing (SMP) may have N times the computation power, but the memory bandwidth usually does not scale up N times. Quite often, the original memory path is shared by multiple processors and performance degradation may be observed when they compete for the shared memory bandwidth.
- Many other common problems affecting the final speedup in parallel computing also apply to OpenMP, like load balancing and synchronization overhead.
- Compiler optimisation may not be as effective when invoking OpenMP. This can commonly lead to a single-threaded OpenMP program running slower than the same code compiled without an OpenMP flag (which will be fully serial).

CHAPTER 6

EVALUATION STUDY

In our experiment we have only implemented the serial wavelet tree and parallel wavelet tree indexing approaches as the further work of BTree and Hash Tree has already being done by many peoples. Results are shown in [Table-2] and [Fig-7,8,9].

In our experiment we have used :-

IDE – DevCpp Library – OpenMP

Processor – i3,i5 and i7

For the data for our work, we have created numerous text file of various sizes. After that, we ran the algorithm on all the above processors to compare the result of the every processor.At the end, following results were found.

Number of Words	Intel i3 processor		Intel i5 processor		Intel i7 processor	
	Parallel algorithm (sec)	Serial algorithm (sec)	Parallel algorithm (sec)	Serial algorithm (sec)	Parallel algorithm (sec)	Serial algorithm (sec)
340197	258.4	269.4	137.2	141.3	148.3	134.5
664237	644.6	731.1	337.6	343.2	388	339.7
989586	978.6	984	511.9	530.8	606.9	598
1324623	1321	1322	700.2	729.4	818	840
1645527	1653	1659	875.7	905.6	913.7	970
1997047	2360	2059	1102	1137	1174	1235

Table .6.1 -Table of Time(sec) of Algorithms

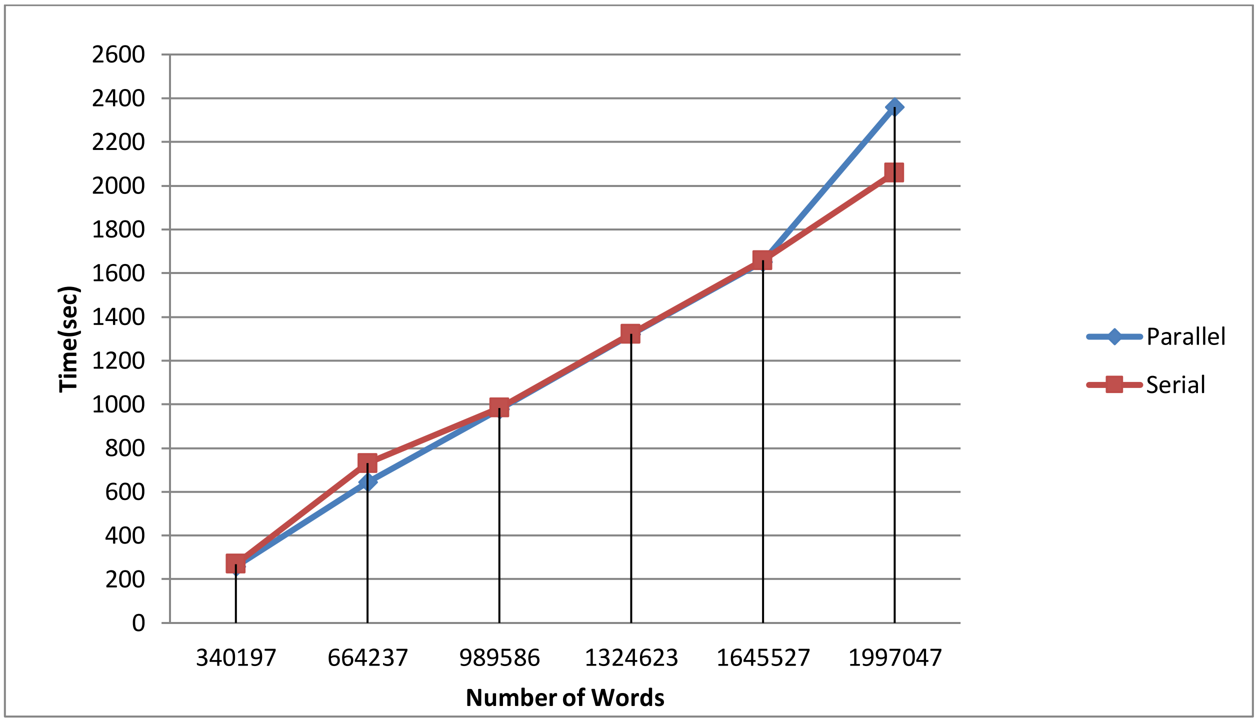


Fig. 6.1- Words VS Time Graph(Intel i3 Processor)

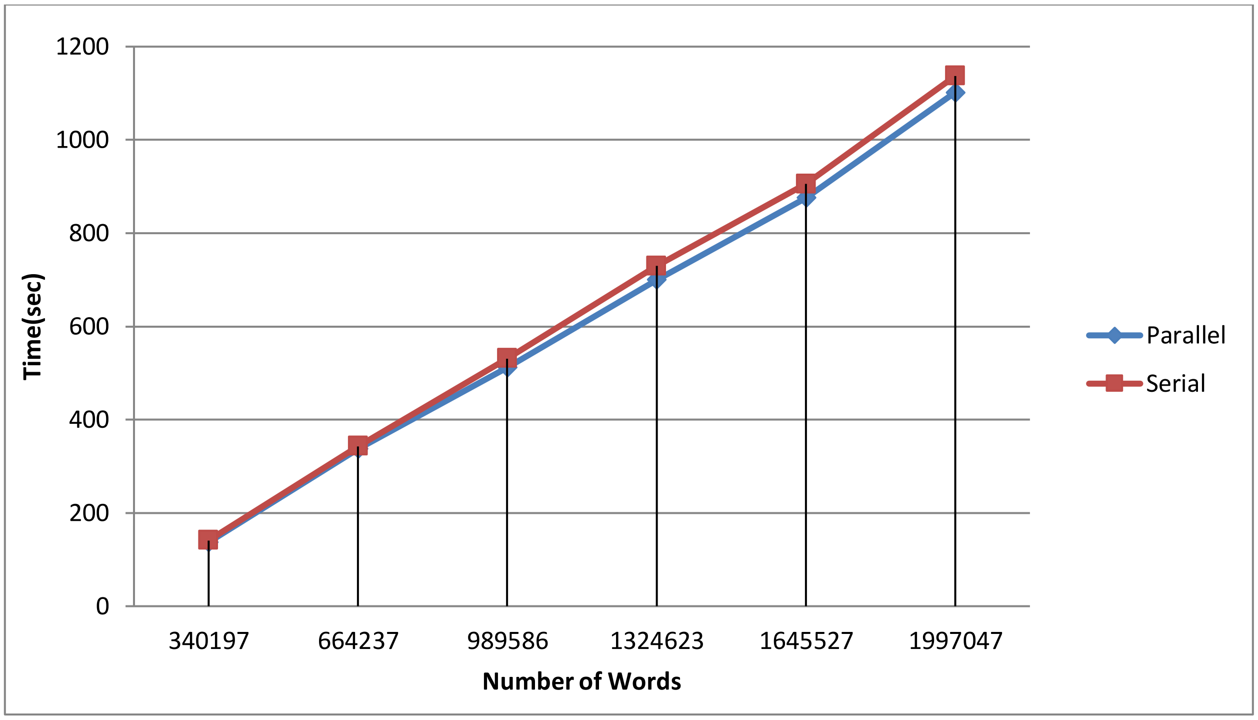


Fig. 6.2- Words VS Time Graph (Intel i5 Processor)

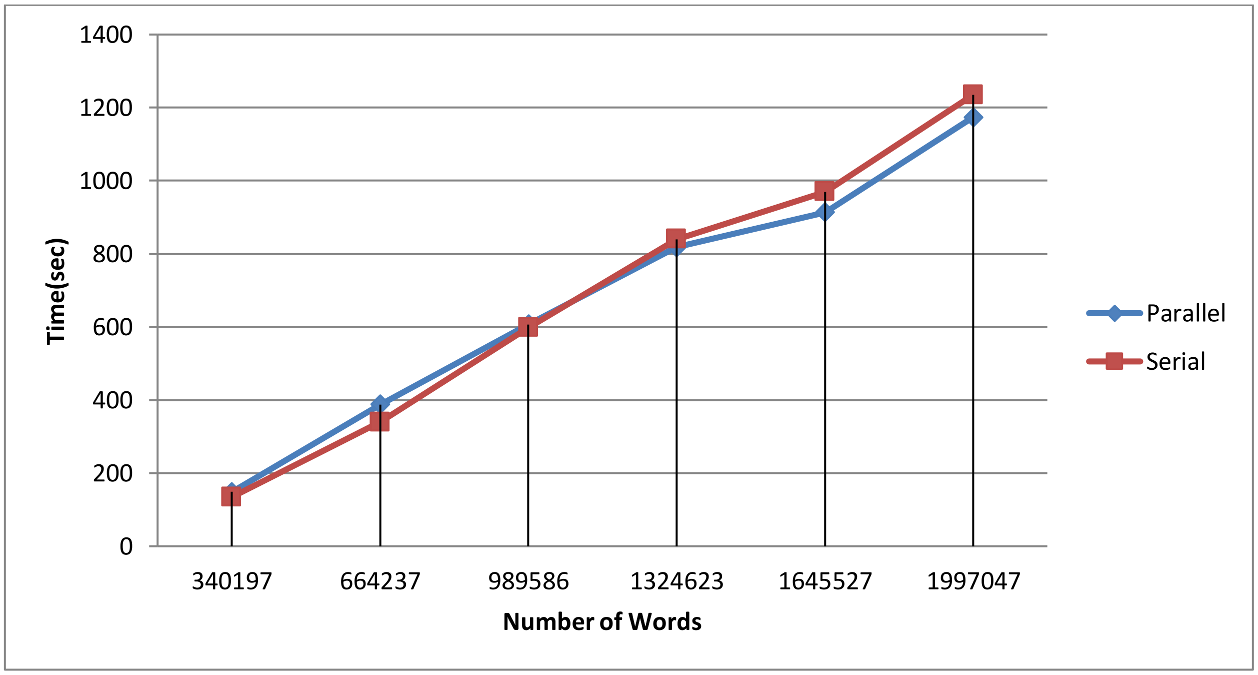


Fig. 6.3- Words VS Time Graph (Intel i7 Processor)

CHAPTER 7

CONCLUSION AND FUTURE WORKS

We have seen the performance comparison or the time complexity analysis of various data structures for textual searching techniques, like b-tree tree, wavelet tree, hash tree, parallel wavelet tree.

The comparison is based upon the time complexity and space complexities of the various data structures used. It is to be noted that the comparison is based on the creation time and space of these data structures.

Wavelet tree is observed to be performing better with increasing query length as compared to other data structures, both in terms of time complexity as well as space complexity.

We have used an existing parallel wavelet tree algorithm proposed by Julian Shun in his paper to performed indexing and has compared the result of the our proposed indexing technique with the existing sequential wavelet tree indexing on various sizes of the document. Also we have theoretically compared the time complexity of the sequential, parallel wavelet tree and various techniques. The experimental results show that the proposed algorithm performed better than the sequential algorithm and consumes less memory than sequential one.

In all it can be said that the parallel extension to already existing wavelet tree is a good step for decreasing the time complexity. We used this parallel wavelet tree for creating inverted index. Main motive behind this was to use less amount of memory but faster access at the same time.

Moving on to the future work that can be added, there are numerous possibilities for improvement of this implementation

These could be as follows:

1. Extending language support
2. Advanced NLP over query string
3. User location based optimized results
4. Browser plugin for ranking optimization
5. Processing log data using big data technologies

Extending language support: The current implementation of search engine primarily deals with English language Testing and support of Hindi language and other regional languages can be added to further enhance the usability of the search engine.

Advanced NLP over query string: Things like sentiment analysis, spelling correction, text classification, named entity recognition, semantics analysis etc, could be implemented so as to further improve the results.

User location based optimized results: Using the current location of user by the help of modern HTML5 geolocation feature and any location-enabled browser, the overall results could be further improved as more personalized and relevant results would come up for each individual user. This concept is also known as locality of reference.

Browser plugin for ranking optimization: A browser plugin could be made to keep track of which documents are more relevant to users and thus their page rank should improve over time. The relevance could be determined by serving that how longer opens on document (webpage), Merging multiple timer would improve results greatly and it

REFERENCES

1. S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. Proc. 7th WWW Conference, 1998.
2. R. Bacza-Yates and B. Ribeiro-Neto. Modern Information Retrieval. Addison Wesley, New York, NY, USA, 1999
3. Search Engine Indexing. https://en.wikipedia.org/wiki/Search engine indexing
4. Design of a Novel Incremental Parallel WebCrawler, Divaker Yadav.
5. .Kuang-huaChen:Indexingand Abstracting http://www.lis.ntu.edu.tw/-khchen/course/ia /slide/ia200401.pdf.
6. S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. Proc. 7th WWW Conference, 1998.
7. The Overview of Web Search Engines, Sunny Lam, February 9, 2001
8. S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. Proc. 7th WWW Conference, 1998
9. DBMSindexing. TutorialsPoint. http://www.tutorialspoint.com/dbms/dbms indexing.htm.
10. Gerald J. Kowalski, Mark T. Maybury Information Storage and Retrieval Systems Theory and Implementation. Second Edition. Kluwer Academic Publication. Chapter-3
11. S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. Proc, 7th WWW Conference, 1998
12. Concept and Purpose of Indexing.http://notes.tyrocity.com/concept-and-purpose of-indexing
13. Search Engine Indexing. https://en.wikipedia.org/wiki/Search engine_indexing

14. Kuang-hua Chen Indexing and Abstractinghttp://www.lis.ntu.edu.tw/-khchen/ course/ia/slide/ia200401.pdf.

15. Mark Sanderson, W. Bruce Croft: The History of Information Retrieval Research. ciir-publications.

16. C. N. Mooers, 'The theory of digital handling of non-numerical information and its implications to machine economics', in Association for Computing Machinery Conference, Rutger University, 1950.

17. Singhal. Amit (2001). "Modem Information Retrieval: A Brief Overview" (PDF). Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 24 (4): 35-43.

18. Mark Sanderson & W. Bruce Croft (2012). "The History of Information Retrieval Research". Proceedings of the IEEE 100: 1444 1451. doi:10.1109/iproc.2012.2189916

19. JE Holmstrom(1948) . "Section III . Opening Plenary Session" . The Royal Society Scientific Information Conference , 21 June-2 July 1948 : report and papers submitted: 85.

20. J. J. Rocchio, 'Relevence Feedback in Information Retrieval' ,Harvard University, ISR-9, 1965.

21. Science Daily - https://www.sciencedaily.com/terms/web_crawler.html

22. https://en.wikipedia.org/wiki/Web_crawler

23. "Automatic Indexing for Research Papers Using References"- Wei Liu, Institute of Scientific and Technical Information of China

24. "Context Based Web Indexing for Semantic Web" -   Anchal Jain, Nidhi Tyagi Lecturer(JPIEAS) Asst. Professor(SHOBHIT UNIVERSITY).

25. "Evaluating the retrieval effectiveness of Web Search engines using a representative query sample" - Dirk Lewandowski Hamburg University of Applied Sciences, Department of Information, Finkenau 35, D--2208 Hamburg, Germany.

26. Williams, R. N. 1991a. Adaptive Data Compression, Norwell. MA: Kluwer Academic. 65.99-101.

27. Effient Single Pass Index Construction for Text Databases Sten Heinz Justin

Zobel.

28. Gonnet, G. (1987). PAT 3.1: An Efficient Text Searching System. User's Manual UW Centre for the New OED, University of Waterloo.

29 T. Johnson (1999). "Performance Measurements of Compressed Bitmap Indices" In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduricz, Stanley B. Zdonik Michael L. Brodie. VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK (PDF). Morgan Kaufmann. pp. 278-89. ISBN 1-55860-615-7.

30. Suflix arrays: A new method for on-line string searches Udi Manber Gene Myers

31. C. Faloutsos and S. Christodoulakis, Signature Files An Access Method for Documents and its Analytical Performance Evaluation, ACM Trans on Office Information Systems (TOOIS), 2, 4, pp. 267-288, Oct. 1984.

32. Haskin, R.L. "Special Purpose Processors for Text Retrieval." Database Engineering, vol. 4, no. 1. pp. 16-29. Sept. 1981.

33.Christodoulakis 1987 "analysis and retrieval of records and objects using optical disk technology"

34. MOOERS. C. 1949. Application of Random Codes to the Gathering of Statistical Information." Bulletin 31. Zator Co., Cambridge, Mass. Based on M.S. thesis, MIT. January 1948.

35. LARSON, R. 1996, Geographic information retrieval and spatial browsing, Inc

and Libraries: Patrons, Maps and Spatial Information, L. Smith and M. Gluck (Eds.)

(Urbana-Champaign: University of Illinois), pp. 81-124.