COMPARATIVE STUDY ON DATA SEARCHING IN LINKED LIST & B-TREE AND B+TREE TECHNIQUES

AHMED ESHTEWI S GIUMA

A dissertation submitted in partial fulfillment of the requirement for the award of the Degree of Master of Computer Science (Software Engineering)

The Department of Software Engineering
Faculty of Computer Science and Information Technology
Universiti Tun Hussein Onn Malaysia

MARCH 2015
ABSTRACT

There are many methods of searching large amount of data to find one particular piece of information. Such as finding the name of a person in a mobile phone record. Certain methods of organizing data make the search process more efficient. The objective of these methods is to find the element with the least time. In this study, the focus is on time of search in large databases, which is considered an important factor in the success of the search. The goal is choosing the appropriate search techniques to test the time of access to data in the database and what is the ratio difference between them. Three search techniques are used in this work namely; linked list, B-tree, and B+ tree. A comparison analysis is conducted using five case databases studies. Experimental results reveal that after the average times for each search algorithms on the databases have been recorded, the linked list requires lots of time during search process, with B+ tree producing significantly low times. Based on these results, it is clear that searching in B- tree is faster than linked list at a ratio of (1: 5). The searching time in a B+ tree is faster than B- tree at the ratio of (1: 2). The searching time in a B+ tree is faster than linked list at the ratio of (1: 8). With that, it can be concluded that B+ tree is the fastest technique for data access.
ABSTRAK

Terdapat banyak kaedah dalam pencarian suatu maklumat dari satu kumpulan data yang banyak. Contohnya seperti mencari nama dalam telefon bimbit. Sestengah kaedah menguruskan data bagi menjadikan proses pencarian lebih efisien. Objektif kaedah yang dibincangkan adalah untuk mencari data dengan cepat. Dalam kajian ini, tumpuan kajian adalah pada masa carian dalam pengkalan data yang besar dimana ia adalah satu factor penting dalam menentukan kejayaan dalam carian. Matlamatnya adalah memilih teknik yang paling sesuai dalam carian data didalam pengkalan data dan perbandingan dalam peratus masa capaian diantara teknik teknik tersebut. Tiga jenis carian dikaji iaitu linked list, B-tree dan B+ tree satu analisa perbandingan dibuat dengan menggunakan lima kajian kes. Hasil kajian telah laporkan dimana linked list memerlukan banyak masa dalam carian berbanding B+ tree. Berdasarkan keputusan ini telah menunjukkan carian dalam B- tree adalah pantas berbanding linked list dengan kadar (1:5). Carian masa dalam B+ tree adalah lebih baik berbanding linked list dengan kadar (1:2). Sementara itu carian masa dalam B+ tree adalah lebih laju berbanding linked list dengan nisbah (1:8). Dengan itu, dapatlah dirumuskan B+ tree adalah teknik yang paling laju dalam capaian data.
CONTENTS

TITLE  i
DECLARATION  ii
DEDICATION  iii
ACKNOWLEDGEMENT  iv
ABSTRACT  v
ABSTRAK  vi
CONTENTS  vii
LIST OF TABLES  xiii
LIST OF FIGURES  xv

CHAPTER 1 INTRODUCTION  1
  1.1 Background  1
  1.2 Problem Statement  2
  1.3 Project Objectives  3
  1.3 Project Scope  3
  1.4 Outline of the Report  3

CHAPTER 2 LITERATURE REVIEW  5
  2.1 Introductions  5
  2.2 Data Structure of Linked Lists  6
    2.2.1 Searching in Linked List  7
    2.2.2 Advantages and Disadvantages for Linked List  8
CHAPTER 3 RESEARCH METHODOLOGY

3.1 Introduction

3.2 The Proposed Methodology for Comparative Study on Database Speed Searching

3.2.1 Load of Data from the Source

3.2.3 Load Data to B-Tree

3.2.3 Calculate Searching Time Using B-Tree

3.2.4 Load Data to Linked List

3.2.5 Calculate Searching Time Using Linked List

3.2.6 Load Data to B+Tree
3.2.7 Calculate Searching Time Using B+tree

3.2.8 Comparative Study between Linked List & B-Tree and B+ Tree

3.2.9 Calculate the Result

3.3 Performance Measure

3.4 Chapter Summary

CHAPTER 4 IMPLEMENTATION AND DISCUSSIONS OF RESULTS

4.1 Introduction

4.1.1 The Complexity

4.1.1.1 The Complexity of Using the Link List Algorithm

4.1.1.2 The Complexity of Using the B-Tree Algorithm

4.1.1.3 The Complexity of Using the B+Tree Algorithm

4.1.2 The Inputs

4.1.3 The Queries

4.2 Data Source

4.2.1 Data Source for First Case Study (The World)

4.2.2 Data Source for Second Case Study (Employees)

4.2.3 Data Source for Third Case Study (Immigration)

4.2.4 Data Source for Fourth Case Study (Libyana)

4.2.5 Data Source for Fifth Case Study (Staff)

4.2.6 The Size Difference between the Databases

4.3 Load of Data from the Source

4.3.1 Load of World Database from the Source

4.3.2 Load of Employees Database from the Source
4.3.3 Load of Immigration Database from the Source 44
4.3.4 Load of Libyana Database from the Source 45
4.3.5 Load of Staff Database from the Source 47

4.4 Calculate Searching Time Using World Database 48
4.4.1 Calculate Searching Time Using Linked List 48
4.4.2 Calculate Searching Time Using B-Tree 50
4.4.3 Calculate Searching Time Using B+Tree 52

4.5 Calculate Searching Time Using Employees Database 53
4.5.1 Calculate Searching Time Using Linked List 53
4.5.2 Calculate Searching Time Using B-Tree 55
4.5.3 Calculate Searching Time Using B+Tree 57

4.6 Calculate Searching Time Using Immigration Database 58
4.6.1 Calculate Searching Time Using Linked List 58
4.6.2 Calculate Searching Time Using B-Tree 60
4.6.3 Calculate Searching Time Using B+Tree 62

4.7 Calculate Searching Time Using Libyana Database 63
4.7.1 Calculate Searching Time Using Linked List 63
4.7.2 Calculate Searching Time Using B-Tree 65
4.7.3 Calculate Searching Time Using B+Tree 67

4.8 Calculate Searching Time Using Staff Database 68
4.8.1 Calculate Searching Time Using Linked List 68
4.8.2 Calculate Searching Time Using B-Tree 70
4.8.3 Calculate Searching Time Using B+Tree 72
4.9 Results Discussions

4.9.1 Search Time Using World Database Case Study

4.9.1.1 The Application of Linked List on the World Database

4.9.1.2 The Application of B-Tree on the World Database

4.9.1.3 The Application of B+Tree on the World Database

4.9.2 Analysis of the Results of the Application of Algorithms on World Database

4.9.3 Search Time Using Employees Database Case Study

4.9.3.1 The Application of Linked List on the Employees Database

4.9.3.2 The Application of B-Tree on the Employees Database

4.9.3.3 The Application of B+Tree on the Employees Database

4.9.4 Analysis of the Results of the Application of Algorithms on Employees Database

4.9.5 Search Time Using Immigration Database Case Study

4.9.5.1 The Application of Linked List on the Immigration Database

4.9.5.2 The Application of B-Tree on the Immigration Database

4.9.5.3 The Application of B+Tree on the Immigration Database

4.9.6 Analysis of the Results of the Application of Algorithms on Immigration Database
4.9.7 Search Time Using Libyana Database Case Study 85
  4.9.7.1 The Application of Linked List on the Libyana Database 85
  4.9.7.2 The Application of B-Tree on the Libyana Database 86
  4.9.7.3 The Application of B+Tree on the Libyana Database 87

4.9.8 Analysis of the Results of the Application of Algorithms on Libyana Database 87

4.9.9 Search Time Using Case Study Staff Database 89
  4.9.9.1 The Application of Linked List on the Staff Database 89
  4.9.9.2 The Application of B-Tree on the Staff Database 90
  4.9.9.3 The Application of B+Tree on the Staff Database 91
  4.9.10 Analysis of the Results of the Application of Algorithms on Staff Database 91

4.11 Chapter Summary 93

CHAPTER 5 CONCLUSIONS 94
  5.1 Objectives Achievement 94
  5.2 Conclusion 94
  5.3 Future Work 95

REFERENCES 96

APPENDIX 98

VITA 156
LIST OF TABLES

4.1 Complexity Connected to the Database and Load Data 27
4.2 Complexity of Linked List Algorithm 28
4.3 Complexity Connected to the Database and Load Data 29
4.4 Complexity of B-Tree Algorithm 30
4.5 Complexity Connected to the Database and Load Data 31
4.6 Complexity of B+ Tree Algorithm 32
4.7 Specifications for the Database World 35
4.8 Specifications for the Database Employees 36
4.9 Specifications for the Database Immigration 37
4.10 Specifications for the Database Libyana 38
4.11 Specifications for the Database Staff 39
4.12 The Size of the Databases 40
4.13 Application of Linked List on the World Database Results 74
4.14 Application of B-Tree on the World Database Results 75
4.15 Application of B+Tree on the World Database Results 75
4.16 The Results of the Application of Algorithms on World Database 76
4.17 Application of Linked List on the Employees Database Results 78
4.18 Application of B-Tree on the Employees Database Results 79
4.19 Application of B+Tree on the Employees Database Results 79
4.20 The Results of the Application of Algorithms on Employees Database 80
4.21 Application of Linked List on the Immigration Database Results 82
4.22 Application of B-Tree on the Immigration Database Results 82
4.23 Application of B+Tree on the Immigration Database Results 83
4.24 The Results of the Application of Algorithms on Immigration Database 84
4.25 Application of Linked List on the Libyana Database Results 86
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.26</td>
<td>Application of B-Tree on the Libyana Database Results</td>
<td>85</td>
</tr>
<tr>
<td>4.27</td>
<td>Application of B+Tree on the Libyana Database Results</td>
<td>85</td>
</tr>
<tr>
<td>4.28</td>
<td>The Results of the Application of Algorithms on Libyana</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td>Database</td>
<td></td>
</tr>
<tr>
<td>4.29</td>
<td>Application of Linked List on the Staff Database Results</td>
<td>90</td>
</tr>
<tr>
<td>4.30</td>
<td>Application of B-Tree on the Staff Database Results</td>
<td>90</td>
</tr>
<tr>
<td>4.31</td>
<td>Application of B+Tree on the Staff Database Results</td>
<td>91</td>
</tr>
<tr>
<td>4.32</td>
<td>The Results of the Application of Algorithms on Staff</td>
<td>92</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

2.1 Linked Lists .................................................. 6
2.2 The Name of a Linked List Versus the Names of Nodes ................................. 7
2.3 Search Algorithm for Linked List ................................ 8
2.4 B- Tree Work .................................................. 10
2.5 B-Tree Search Algorithm ......................................... 13
2.6 B+Tree Search Algorithm ......................................... 15
3.1 Steps Involved in the Research .................................... 20
3.2 Distribution of Database Load of Data ................................ 21
4.1 Authentication Required ............................................. 33
4.2 Data Source ........................................................ 34
4.3 Database World Specifications ........................................ 35
4.4 Database Employees Specifications .................................... 36
4.5 Database Immigration Specifications .................................... 37
4.6 Database Libyana Specifications ....................................... 38
4.7 Database Staff Specifications ........................................... 39
4.8 The Difference in Size between Databases .............................. 40
4.9 Load of World Database from the Source by Using the Country Code .......... 41
4.10 Load of World Database from the Source .................................. 42
4.11 Load of Employees Database from the Source by Using the Employee Number ........................................................................ 43
4.12 Load of Employees Database from the Source .................................. 43
4.13 Load of Immigration Database from the Source by Using the Id No ........... 44
4.14 Load of Immigration Database from the Source .................................. 45
4.15 Load of Libyana Database from the Source by Using the Phone Number ........................................................................ 46
4.16 Load of Libyana Database from the Source .................................. 46
4.17 Load of Staff Database from the Source by Using the Id Number .................. 47
4.18 Load of Staff Database from the Source
4.19 Load a Copy of the World Database to Linked List
4.20 How to Search by Linked List and View Searching Time
4.21 Load a Copy of the World Database to B-Tree
4.22 How to Search by B-Tree and View Searching Time
4.23 Load a Copy of the World Database to B+Tree
4.24 How to Search by B+Tree and View Searching Time
4.25 Load a Copy of the Employees Database to Linked List
4.26 How to Search by Linked List and View Searching Time
4.27 Load a Copy of the Employees Database to B-Tree
4.28 How to Search by B-Tree and View Searching Time
4.29 Load a Copy of the Employees Database to B+Tree
4.30 How to Search by B+Tree and View Searching Time
4.31 Load a Copy of the Immigration Database to Linked List
4.32 How to Search by Linked List and View Searching Time
4.33 Load a Copy of the Immigration Database to B-Tree
4.34 How to Search by B-Tree and View Searching Time
4.35 Load a Copy of the Immigration Database to B+Tree
4.36 How to Search by B+Tree and View Searching Time
4.37 Load a Copy of the Libyana Database to Linked List
4.38 How to Search by Linked List and View Searching Time
4.39 Load a Copy of the Libyana Database to B-Tree
4.40 How to Search by B-Tree and View Searching Time
4.41 Load a Copy of the Libyana Database to B+Tree
4.42 How to Search by B+Tree and View Searching Time
4.43 Load a Copy of the Staff Database to Linked List
4.44 How to Search by Linked List and View Searching Time
4.45 Load a Copy of the Staff Database to B-Tree
4.46 How to Search by B-Tree and View Searching Time
4.47 Load a Copy of the Staff Database to B+Tree
4.48 How to Search by B+Tree and View Searching Time
4.49 Time of Each Search Algorithms for Executing Queries for World Database
4.50 Time of Each Search Algorithms for Executing Queries for
4.51 Time of Each Search Algorithms for Executing Queries for Immigration Database

4.52 Time of Each Search Algorithms for Executing Queries for Libyana Database

4.53 Time of Each Search Algorithms for Executing Queries for Staff Database
CHAPTER 1

INTRODUCTION

1.1 Background

Data is defined as a set of valuable information with certain similarities, which is usually sorted in such way where it may be easily retrieved by other relevant parties. The Internet or a library is a storage facility providing avenue for the accessibility of data, and such storages are known as databases. Every organization deals with a series of databases respectively. For instance, the police may have a database of criminal records, where a car showroom would have a database of vehicle history. The size of the database directly affects the effectiveness in searching the data. Thus, every data should be traced via a database, based on the following criteria:

(i) Ability to search for a specific item.
(ii) Ability to search for related items to a known item.
(iii) Ability to search in a specific field or fields.
(iv) Ability to combine search terms using Boolean logic.

The most noticeable problem in the world of computer science and information technology would be the storage and retrieval of data. There are applications and search engines which are capable to access a large virtual database in a short period of time. Nevertheless, the scope of the hits on the desired data might be large, to an extent that the user still cannot find what he/she is looking for. However, there are certain infrastructures applicable for retrieval of data efficiently. The most common search structure would be the multi way balanced B-tree. As the name suggests, it consists of leaf and internal, or also known as the nodes. The
internal nodes are basically the trace index to the leaf nodes, whereas the leaf nodes are the data carrier. As for this infrastructure is by far the most effective method in the maintenance of disk data (Askitis et al., 2009).

Other search structures exist namely, the linked list and the B+ tree described in the following paragraphs.

In the context of computer science, linked list are a structured data, used in retrieval of sequential objects, allowing flexibility to add or remove intermediate elements in the sequence. Instead of having a series of arrays, linked list consists of nodes, that stores value and reference of the next node. Though the insertion and removal of nodes are fast, the access to the elements could be slow since in order to access node ten, the link would go through the first nine nodes if no removals were made. Random access elements on the other hand are accessed arbitrarily (John Wiley & Sons, 2010).

A B+ tree consists of a root, which may be a leaf or a node with more than two children, in where the actual number of children for a node is denoted as \( m \). The root is an exception. The primary value of a B+ tree is in the stored data for efficient retrieval in a block-oriented storage context such as the file systems. Unlike the binary search trees, B+ trees have high fan outs or pointers to children nodes in a certain node (Navathe et al., 2010).

1.2 Problem Statement

One of the problems that faces large databases users is the noticeable lateness of data retrieval which can lead to boredom and the loss of user's time by waiting for the completion of data access and retrieval process. In order to minimize the searching time and the loss of the data, many of the programmers and developers of software engineering development have designed several techniques that can help to increase the searching speed and also provide a good compromise for databases users. Developers have developed many of the algorithms that do the searching process and all the work to achieve the fastest time in the data retrieval process. But there is a difference between these algorithms in terms of speed, there are high-speed algorithms and other medium-speed and slow speed. That make databases designers find it difficult to determine which algorithm is faster. Because of that researchers
have compared between many of the techniques used in order to determine the fastest technique and facilitate the selection of any appropriate algorithm in the search process. In this research comparative study will be conduct on the three algorithms (linked list, B-tree and B + tree) to determine the fastest and also to determine the percentage difference between the three algorithms. In this study research five different sized databases will be used in order to get more accurate results.

1.3 Project Objectives

The objectives of this research are summarized as follow:

(i) To develop and implement linked list, B-tree and B+tree by using one of the programming languages.
(ii) To compare the three proposed techniques using the five case studies depending on the different sizes of the data.
(iii) To evaluate and analysis results based on time and identify any faster technique, and calculate the amount of the difference between them.

1.3 Project Scope

This research focuses on the problem of time search in databases. Therefore, linked list, B-tree and B+tree techniques will be used to test the speed of access to data in the database and will be compared using the five case studies.

1.4 Outline of the Report

This research consists of five chapters. Chapter 1 is an overview of the project and the main objectives of the project. It consists of the scope of work covered and the project’s objectives. Chapter 2 illustrates the literature review of the project. It also gives a brief explanation in general information about automated testing for database system in this project. Chapter 3 discusses the methodology used to obtain the entire objectives of this project and tools. Chapter 4 explains the implementation and the
detailed steps in this work as well as the results and discussion. Chapter 5 includes
the objectives achieved, disadvantages, future work, and conclusion of the project.
CHAPTER 2

LITERATURE REVIEW

2.1 Introductions

Historically, memory limit was restricted, so extensive information accumulations must be put away on databases, which utilize information structures, for example, linked list and B-trees. With the accessibility of expansive memories, this confinement has been loose. Correspondingly, various new requisitions have risen in such fields as bio-informatics and computational semantics that oblige looking immense accumulations in memory. A B-tree-like information structure implicit memory is still a great answer for such issues (Helen, 2011).

Nodes are arranged in a certain way that they communicate sequentially in a linked list. In a basic structure, under the least complex structure, every previous node acts as a predecessor of the current node, and every current node acts as a successor of the previous node. Removal and addition of nodes are dynamic, where it could be done from any point in the list.

Connected records are easily comparable as they store information beneficial to the customer. A similar structure of connected records would store the similar type of data. The interchange methodologies and the functionality of connected records would be a good research to conduct on (Nick, 2010).

A linked list stockpiling is effective in such way that a client does not have to worry about the relevancy of data acquired. Linked list rundown information stockpiling is where the information are retrieved haphazardly. The incorporation of
linked list in corresponding channels, organization of binary trees, stack building, queues in programming, and overseeing social databases creates an ease in access.

The exhibits are the most widely recognized information structure used to store data. Mostly, clusters are helpful in terms of linguistic assistance in getting to any component via its record number (Nick, 2010).

B-tree is a tree information structure that keeps information sorted, where logarithmic insertions and cancellations are easy. The B-tree is a generalization of a binary inquiry tree in that a node can have more than two branches. Unlike the common tree structures, the B-tree have improved framework and composes numerous information. It is commonly used in databases and document frameworks. It is an effective method in placing and retrieving records in a database. However, the significance of the alphabet B has not been theoretically expressed. The B-tree calculation saves time since a medium exist to run through the existing records, with a fast moving algorithm (Margaret, 2009).

2.2 Data Structure of Linked Lists

Linked lists consists of data and link. Via the link, each data element contains location information about the next immediate element. The index name is basically the pointer variable name in the linked list. The following Figure 2.1 illustrates a linked list, addressed as scores, which consists of four elements. An example of an empty linked list, or a null pointer is shown in Figure 2.1.

![Figure 2.1: Linked Lists (Behrouz & Firouz, 2008)](image-url)
Each linked list should be named in such way that it could be differentiated from the elements and the nodes itself. Figure 2.2 displays the name of a selected linked list, which is the head pointer that directs the link to the first node in the linked list. A node would only have implicit rather than explicit name (Behrouz & Firouz, 2008).

![Figure 2.2: The Name of a Linked List Versus The Names of Nodes](image)

2.2.1 Searching in Linked List

Two separate pointers, known as previous (pre) and current (cur) are used in nodes. In the initial stage of a search, the pre pointer would be null, whereas the cur pointer would be linked to the first node of the link. The algorithm of this search structure links these two pointers all the way towards the end of the list. If the target value is bigger than the values in the entire list, the movement of the pointers would be slow. Figure 2.3 illustrates a linked list search algorithm with the pre and cur pointers (Behrouz & Firouz, 2008).
**Search algorithm for linked list**

Algorithm: Search linked list (target, list)

Purpose: Search the list using two pointers: per and cur

Post: None

Per: The linked list (head pointer) and target value

Return: the position of per and cur pointers and the value of the flag (true or false)

```
{ 
    Per ← null
    The previous value = null
    Cur ← list
    Current value
    While (target < (*cur).data )
    {
        Per ← cur
        Cur ← (*cur).link
    }
    If the Current value = flag = true
    If ((*cur).data=target) flag ← true
    Else flag ← false
    Return (cur, per, flag)
}
```

Figure 2.3: Search Algorithm for Linked List (Behrouz & Firouz, 2008)

### 2.2.2 Advantages and Disadvantages for Linked List

In simple terms, linked lists are a basic chain containing nodes or data, linked via pointers that points the current data towards the next data.

#### 2.2.2.1 Advantages

All data linked in the list are from the similar group or search field. These are some advantages of linked list:

(i) The information structure consumes low external memory during run time as it is a real time system.
(ii) The addition or removal of a node from the list are considerably simple.

(iii) The stacking and queuing of data is easy, resulting in linearity of structure.

(iv) No time delays are faced since the access time of data in the list is extremely fast.

### 2.2.2.2 Disadvantages

Some of the visible disadvantages of linked lists are:

(i) Since pointers require additional capacity memory, a memory wastage occurs.

(ii) The server client does not have access to the linked list, these all nodes should be effectively provided during a search.

(iii) Massive amount of time required in joined rundown, since distinctive nodes are not separated during adjoining of memory allotments.

(iv) Reverse crossing is difficult in interfaced rundown. In an independently joined rundown, it is difficult to navigate through linked list. In doubly linked, it is easier to intercept from the end of link, and also providing storage capacity for the back pointer.

### 2.2.3 Implementing Linked Lists

Linked lists are used to sort resources in a certain required manner, independent of the memory address each record is allocated. These information are numerically created via the ID number, and sorted via name. It joins relevant records to fulfill the search field.

Many developers still utilize linked lists as their infrastructure foundation. Record connection is an interesting field to look deeply into as:

(i) The straightforwardness in linked list structure. All operations such as omitting an inserting schedules are easy due to the structure of joined rundowns.

(ii) There is no complexity in the algorithm. The algorithm calculation and pointer concentration could be designed as many ways by the developers to cater for the clients needs.
(iii) Pointer intensive linked rundown issues are due to the pointers themselves. Structure if the lists are pointer concentrated. The calculations disconnects and reconnects the pointers successfully. The connection of the records puts a developers grasp towards pointers to a test.

(iv) Visual is an important word in the context of programming. A software engineer would visualize the functionality of his algorithm in a clients point of view. Perl and java utilizes layered and reference based information structure that is easily visualized. In record connection, joined rundown could be visualized in terms of connecting the nodes (Nick Parlante, 2010).

2.3 Data Structure of B-Tree

B-trees are favored when choice focuses, called nodes, are on hard plate as opposed to in arbitrary access memory (RAM). It takes many times longer to get to an information component from hard circle as contrasted and getting to it from RAM, in light of the fact that a plate drive has mechanical parts, which read and compose information much more gradually than simply electronic media. B-trees spare time by utilizing nodes with numerous extensions (called kids), contrasted and two fold trees, in which every node has just two youngsters. At the point when there are numerous youngsters for every node, a record could be found by passing through fewer nodes than if there are two kids for every node. A disentangled illustration of this guideline is indicated below in Figure 2.4.

![Figure 2.4: B-Tree Work (Margaret, 2009)](image-url)
In a tree, records are put away in areas called takes off. This name infers from the way that records dependably exist at end focuses; there is nothing past them. The most extreme number of kids for every node is the request of the tree. The amount of the obliged plate which it gets to is the profundity. The picture on the left shows a paired tree placing a specific record in a set of eight clears out. The picture on the right shows a B-tree of request three finding a specific record in a set of eight leaves. The parallel tree at left has a profundity of four; the B-tree at right has a profundity of three. Plainly, the B-tree permits a wanted record to be placed speedier, expecting all other framework parameters to be indistinguishable. The tradeoff is that the choice methodology at every node is more convoluted in a B-tree as contrasted and a double tree. A refined system is obliged to execute the operations in a B-tree. Nevertheless this project is put away in RAM, so it runs quick.

In a down to earth B-tree, there might be thousands, millions, or billions of records. Not all leaves essentially hold a record, yet in any event a large portion of them do. The distinction in profundity between twofold tree and B-tree plans is more amazing in a useful database than in the case delineated here, on the grounds that certifiable B-trees are of higher request (32, 64, 128, or more). Contingent upon the number of records in the database, the profundity of a B-tree can and regularly does change. Including a huge enough number of records will expand the profundity; erasing a vast enough number of records will diminish the profundity. This guarantees that the B-tree works ideally for the amount of records it holds (Margaret, 2009).

2.3.1 Advantages and Disadvantages for B-Tree

B-trees are powerful not just because they allow any file item to be immediately located using any attribute as a key, but because they work even when the file is very dynamic.

2.3.1.1 Advantages

B-Trees take advantage of this by maintaining a balanced tree structure through the use of files:
(i)  Keeps keys in sorted request for consecutive crossing.
(ii) Uses a various leveled file to minimize the amount of plate peruses .
(iii) Uses in part full squares to speed insertions and cancellations .
(iv)  Keeps the record adjusted with an exquisite recursive calculation .
(v)   In expansion, a B-tree minimizes squander by verifying the inside nodes at any rate half full. A B-tree can deal with a self-assertive number of insertions and cancellations.

2.3.1.2 Disadvantages

The B-tree is not without disadvantages that hinder the search process within the system, including:
(i)  For information incorporating all out variables with distinctive number of levels, data pick up in choice trees are inclined to be energetic about those qualities with more levels.
(ii) Calculations can get exceptionally intricate especially if numerous qualities are indeterminate and/or if numerous results are joined.
Searching an uneven tree may oblige navigating a subjective and flighty number of nodes and pointers.

2.3.2 Implementing B-Tree

B-tree is a good information structure for putting away enormous measures of information for quick recovery. When there are millions and billions of things in a B-tree, this is the point at which they have fun. B-trees are generally a shallow yet wide information structure. While different trees can develop high, a common B-tree has a solitary digit stature, even with millions of entries.

2.3.3 Searching in B-Tree

B-tree search takes as input a pointer to the root node x of a sub tree and a key k to be searched for in that sub tree. The top-level call is thus of the from B-tree search
(root[T], k). If k is in the B-tree, B-tree search returns the ordered pair (y, i) consisting of a node y and an index i such that key i[y] = k. Otherwise, the value NIL is returned. Figure 2.5 shows an B-tree Search Algorithm (Thomas, 2009).

```plaintext
function B-TREE-SEARCH(x, k)
    returns (y, i) such that key i[y] = k or NIL
    i ← 1
    while i ≤ n[x] and k > key i[x]
        do i←i + 1
    if i ≤ n[x] and k = key i[x]
        then return (x, i)
    if leaf[x]
        then return NIL
    else DISK-READ(ci[x])
    return B-TREE-SEARCH(ci[x], k)
```

Figure 2.5: B-Tree Search Algorithm (Lefteris & Dani, 2013)

### 2.4 Data Structure of B+Tree

In computer science, a tree is a widely used data structure. A data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently to simulate a hierarchical tree structure with a set of linked nodes.

A B+ tree is a type of tree which represents sorted data in a way that allows efficient insertion, retrieval and removal of records each of which is identified by a key. It is a dynamic, multilevel index, with maximum and minimum bounds on the number of keys in each index segment usually called a "block" or "node". In contrast to a B-tree, all records are stored at the leaf level of the tree; only keys are stored in interior nodes (Prabhakar & Vineet, 2010).
2.4.1 Advantages and Disadvantages for B+Tree

The B+tree is a modification of the B-tree that stores data only in leaf nodes, minimizing search cost in the common and worst case, and (optionally) links together all the leaf nodes in a linked list, optimizing ordered access.

2.4.1.1 Advantages

There are various advantages and benefits which B+ tree possessed to assist in search process within the system. This includes:

(i) B+ tree able to provide a reasonable performance for direct access.
(ii) B+ tree able to provide an great performance for sequential and accesses in range.
(iii) B+ able to perform the searching process faster compared to others.
(iv) The potential of B+ tree being a single-dimensional index for emerging and future applications.

2.4.1.2 Disadvantages

However, despite the various advantages over others, B+tree is not a perfect system. It also consists of disadvantages that would affect the search process within the system. These disadvantages are as below:

(i) The insert mechanism in B+tree is more complex than other B-trees.
(ii) The removal/deletion in B+tree is also more complex as compared to other B-trees.
(iii) Wastages of memory space as the search key values are duplicated (Satinder & Aditya, 2009).

2.4.2 Implementing B+ Tree

There are some important incentives in implementing B+ tree:

(i) In B+tree, the searching process is becoming easy.
(ii) B+ trees are able to store the redundant search key.
(iii) At the same time, these trees did not consume much space.

2.4.3 Searching in B+ Tree

The procedure using the B+ tree as the access structure to search for record. These algorithms assume the existence of a key search field, they must be modified appropriately for the case of a B+tree on a non-key filed searching for record with search key field value k, using a B+ tree. A B+ tree, data pointers are stored only at the leaf nodes, therefore the structure of the leaf nodes vary from the structure of the internal (non-leaf) nodes. If the search field is a key field, the leaf nodes have a value for every value of the search field, along with the data pointer to the record or block. If the search field is a non-key field, the pointer points to a block containing pointers to the data file records, creating an extra level of indirection (similar to option 3 for the secondary indexes). The leaf nodes of the B+ Trees are linked to provide order access on the search field to the record. The first level is similar to the base level of an index. Some search field values in the leaf nodes are repeated in the internal nodes of the B+ trees, in order to guide the search. Figure 2.6 shows B+tree search algorithm (Navathe et al., 2010).

```
Function: search (k)
    return tree_search (k, root);
Function: tree_search (k, node)
    if node is a leaf then
        return node;
    switch k do
    case k < k_0
        return tree_search(k, p_0);
    case k_i ≤ k < k_{i+1}
        return tree_search(k, p_{i+1});
    case k_d ≤ k
        return tree_search(k, p_{d+1});
```

Figure 2.6: B+Tree Search Algorithm (Ramez & Shamkant, 2010)
2.5 Related Work

Several researchers have investigated many topics on (search time to exist database) as summarized in the recent survey. While there is a large amount of work related to this dissertation, only the most related topics on generated automated testing have been reviewed and discussed.

Yuxing & Jun (2014) proposed real-time trajectory indexing method based on Mongo DB and mixed with spatio-temporal R-tree, hash table and B-tree for searching leaf nodes. Time in spatio-temporal R-tree is used as another dimension of equal status to space, and a leaf node can only involve a moving object’s consecutive trajectory points. In order to solve the problem of frequent updates and lack of memory, hash table is divided into two kinds: one caches leaf nodes of spatio-temporal R-tree, which are not inserted into spatio-temporal R-tree until they are full or out-dated in the hash table. This improves generation efficiency of real-time trajectory index; the other one caches in-memory nodes which are loaded from external memory, it avoids frequent operations related to external memory. They have build B-tree based on object identification and time in leaf nodes, which benefits trajectory queries for moving objects. In comparison to SETI, the experimental results show that our method has good update efficiency and query performance, and it meets the demand of common trajectory queries in present applications.

Rize & Hyung (2013) proposed a novel B-tree storage scheme, a group round robin based B-tree index storage scheme, which applies a dynamic grouping and round robin techniques for erase-minimized storage of B-tree in flash memory under heavy-update workload. Experiment results show that the proposed scheme is efficient for frequently changed B-tree structure and improves the I/O performance by 2.14X.

Blevins & Jason (2009) proposed A Generic Linked List Implementation in Fortran 95. Develops a standard conforming generic linked list in Fortran 95 which is capable of storing data of any type. The list is implemented using the transfer intrinsic function, and although the interface is generic, it remains relatively simple and minimizes the potential for error. Although linked lists are the focus in the generic programming techniques used are very general and broadly applicable to
other data structures and procedures implemented in Fortran 95 that need to be used
with data of an unknown type.

Braginsky & Erez (2012) presented a design for a lock-free balanced tree,
specifically, a B+tree. The B+tree data structure has an important practical
applications, and is used in various storage-system products. As far as we know this
is the first design of a lock-free, dynamic, and balanced tree, that employs standard compare-and-swap.

Timnat & Shahar (2012) presented design such a linked-list. To achieve
better performance, they have also extended this design using the fast-path-slow-path
methodology. The resulting implementation achieves performance which is
competitive with that of Harris’s lock-free list, while still guaranteeing non-
starvation via wait-freedom. They developed a proof for the correctness and the wait-
freedom of our design.

Timnat & Shahar (2014) presented a transformation of lock-free algorithms
to wait-free ones allowing even a non-expert to transform a lock-free data-structure
into a practical wait-free one. The transformation requires that the lock-free data
structure is given in a normalized from defined in this work. Using the new method,
they have designed and implemented wait-free linked-list, skip list, and tree and we
measured their performance. It turns out that for all these data structures the wait-free
implementations are only a few percent slower than their lock-free counterparts,
while still guaranteeing non-starvation.

Achakeev & Bernhard (2013) proposed the first loading algorithm for MVBT
that meets the lower-bound of external sorting. In addition, their approach is also
applicable to bulk updates. This is achieved by combining two basic technologies,
weight balancing and buffer tree. Their extensive set of experiments confirm the
theoretical findings: their loading algorithm runs considerably faster than performing
updates tuple-by-tuple.
2.6 Chapter Summary

This chapter reviewed the linked list, B-tree and B+tree techniques, its histories and related works regarding linked list, B-tree and B+tree techniques. The next chapter will look into research methodology of the study.
CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

This chapter discusses the suitable methodology to obtain the objectives of this project. There are three methods to be used as the research’s methodology. The methods are linked list, B-tree and B+-tree and they are used for comparative study on database speed searching. The next section discusses the methodology.
3.2 The Proposed Methodology for Comparative Study on Database Speed Searching

![Diagram](image)

Figure 3.1: Steps Involved in the Research

- Method 1: Load Data to B+ Tree
  - Calculate Searching Time
  - Compare Between Linked list & B-tree and B+tree
  - Calculate Result Comparative
  - Results Discussions

- Method 2: Load Data to B- Tree
  - Calculate Searching Time

- Method 3: Load Data to Linked List
  - Calculate Searching Time
Based on Figure 3.1, there are several steps needed for this the comparative study on database speed searching. The first step is to get the database from resources. Then there are three types of search. The first test is the linked list where we withdraw the database and calculate the time taken to search. The second test is to withdraw the database to the B-tree and calculate the time it takes. The third test is using B+ tree where the time taken to withdraw is calculated. Based on the outcomes, a comparison between the tree tools will be carried out to identify any significance.

3.2.1 Load of Data from the Source

The first step of the work of the system is when loading of the data from the source, which are usually very large data. Great time is downloaded into the system and the reason for this is to show the difference in time between the techniques used in the system as shown in Figure 3.2.

![Diagram showing the distribution load of data](image)

Figure 3.2: Distribution Load of Data
3.2.3 Load Data to B-Tree

After the data is raised to the database, it then transferred a copy of this data to the B-tree, as shown in Figure 3.2. The purpose for this is to calculate the time taken when conducting a search process using this technique as the time taken should be lesser than the original time.

3.2.3 Calculate Searching Time Using B-Tree

Time taken to in a data search within the database is calculated using (Big O) concept. The concept is used to measure the complexity of the algorithms used in the search process. In B-tree, the complexity of the algorithm is $O(\log n)$ which increases the search data’s speed thus making the search process more efficient and quicker, which saves a lot of time.

3.2.4 Load Data to Linked List

After the data is raised to the database, it then transferred a copy of this data to the Linked list, as shown in Figure 3.2. The purpose for this is to calculate the time taken when conducting a search process using this technique as the time taken should be lesser than the original time.

3.2.5 Calculate Searching Time Using Linked List

In the linked list technique, the testing time for search is by using (Big O). Using the technology (hash table), linked list technique segments spreadsheets into small units. The search using Big O is conducted where the computational complexity of both singly-linked list and constant-sized hash table is $O(n)$. 
3.2.6 Load Data to B+Tree

In this test the data is raised to the database and loaded to B+tree. The time taken to search is recorded. The time taken should be shorter then original time.

3.2.7 Calculate Searching Time Using B+tree

Are calculated in the time it takes to search for data within the database, using an (Big O) is the concept is to measure the complexity of the algorithms used in the search process. The complexity in B+ Tree is \(O(log_m n)\) which will increase the speed of search for data to become the search process more efficient and more quickly, leading to shortcut a lot of time to search.

3.2.8 Comparative Study between Linked List & B-Tree and B+ Tree

One of the most important stages of the system is when they compare the first, second, and third techniques of the access performance to the queries precisely through the subsets connected with each other. The comparison shall be made by organization direct access or tree structured organization and be an attribute access of one or several attributes to enter.

3.2.9 Calculate the Result

In the final stage of the system, the full results will be shown and the application of all three techniques are illustrated in the forms of tables and charts.

3.3 Performance Measure

There are several factors that may affect the accuracy of the results and should take into account when developing and implementing the program search process used in this research as well as how to evaluate the results of these factors (complexity, input, queries).
The advantage of functions C# in the calculate search time is calculated in
the program.

To calculate the average search time, the following equation is used:

\[
\bar{x} = \frac{1}{N} \sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + \ldots + x_n}{N}
\]

where

\( \bar{x} \): Average.

\( x \): Time taken for search.

\( N \): Number of queries.

3.4 Chapter Summary

This chapter discussed the work of algorithms for linked list, B-tree and B+tree and
how the search process used them to find a difference when searching a large amount
of data. The chapter covered each technique separately and identifies the differences
between them. The next chapter will discuss the results of the system’s implementation.
REFERENCES


