VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELAGAVI – 590018, KARNATAKA

Mini Project Report

on

Used Car Database

Submitted in partial fulfilment of the requirement for the

File Structures Laboratory with mini project [15ISL68]

Bachelor of Engineering

in

Information Science and Engineering

Submitted by

DEEKSHITH M [1JT16IS012]

Under the guidance of Mr.Vadiraja A

Asst.Professor,Department of ISE

Jyothy Institute of Technology

Tataguni, Bengaluru-560082

Jyothy Institute of Technology

Tataguni, Bengaluru-560082

Department of Information Science and Engineering

CERTIFICATE

Certified that the mini project work entitled **Used Car Database** carried out by **Deekshith M[1JT16IS012]** bonafide students of Jyothy Institute of Technology, in partial fulfilment for the award of **Bachelor of Engineering** in **Information Science and Engineering** department of the **Visvesvaraya Technological University, Belagavi** during the year **2019-2020**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library. The mini project report has been approved as it satisfies the academic requirements in respect of mini project work prescribed for the said Degree.

Vadiraja A	Dr. Harshvardhan Tiwari
Guide, Asst. Professor	Associate Professor and HOD
Dept. Of ISE	Dept. Of ISE

External Viva Examiner   Signature with Date:

1.
2.

ACKNOWLEDGEMENT

Firstly, we are very grateful to this esteemed institution **“Jyothy Institute of Technology**” for providing us an opportunity to complete our project.

We express our sincere thanks to our Principal **Dr. Gopalakrishna K for** providing us with adequate facilities to undertake this project.

We would like to thank **Dr. Harshvardhan Tiwari, Associate Professor and Head** of Information Science and Engineering Department for providing for his valuable support.

We would like to thank our guide **Mr. Vadiraja A, Asst. Prof.** for his keen interest and guidance in preparing this work.

Finally, we would thank all our friends who have helped us directly or indirectly in this mini project.

DEEKSHITH M (1JT16IS012)

ABSTRACT

Structural representation of data items in primary memory to do storage & retrieval operations efficiently.

A File Structure allows applications to read, write and modify data. a file system is a method of organizing and retrieving files from a storage medium File systems usually consist of files separated into groups called directories Windows supports three different file systems which are NTFS,FAT32 and exFAT. NTFS is the most modern file system.

Document processing is the generic class for data processing apparatus and corresponding methods for the retrieval of data stored in a database or as computer files. ... Database and file accessing.

This project has been created using Eclipse with the platform of WINDOWS. The project title **“USED CAR DATABASE”** talks about how we process the data accepted from the user.

The purpose of this project is to reduce the time it takes data of the cars which are sold with the help of the programming language and computerized equipment’s.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Introduction to File Structures

In simple terms, a file is a collection of data stored on mass storage (e.g., disk or tape). But there is one important distinction that must be made at the outset when discussing file structures. And that is the difference between the logical and physical organization of the data.

On the whole a file structure will specify the logical structure of the data, that is the relationships that will exist between data items independently of the way in which these relationships may actually be realized within any computer. It is this logical aspect that we will concentrate on. The physical organization is much more concerned with optimizing the use of the storage medium when a particular logical structure is stored on, or in it. Typically for every unit of physical store there will be a number of units of the logical structure (probably records) to be stored in it.

For example, if we were to store a tree structure on a magnetic disk, the physical organization would be concerned with the best way of packing the nodes of the tree on the disk given the access characteristics of the disk.

Like all subjects in computer science the terminology of file structures has evolved higgledy-piggledy without much concern for consistency, ambiguity, or whether it was possible to make the kind of distinctions that were important.

It was only much later that the need for a well-defined, unambiguous language to describe file structures became apparent. In particular, there arose a need to communicate ideas about file structures without getting bogged down by hardware considerations.

1.2 Introduction to JAVA

Like any programming language, the Java language has its own structure, syntax rules, and programming paradigm. The Java language's programming paradigm is based on the concept of OOP, which the language's features support. The Java language is a C-language derivative, so its syntax rules look much like C's. For example, code blocks are modularized into methods and delimited by braces ({and}), and variables are declared before they are used. Structurally, the Java language starts with packages. A package is the Java language's namespace mechanism. Within packages are classes, and within classes are methods, variables, constants, and more.

1.3 Introduction to B-Tree

B-Tree is a self-balancing search tree. In most of the other self-balancing search trees (like AVL and Red Black Trees), it is assumed that everything is in main memory.

To understand use of B-Trees, we must think of huge amount of data that cannot fit in main memory. When the number of keys is high, the data is read from disk in the form of blocks. Disk access time is very high compared to main memory access time.[1]

The main idea of using B-Tree is to reduce the number of disk accesses. Most of the tree operations (search, insert, delete, max, min,etc) require O(h) disk accesses where h is height of the tree.

B-Tree is a fat tree.

Height of B-Tree is kept low by putting maximum possible keys in a B-Tree node. Generally, a B-Tree node size is kept equal to the disk block size.

Since h is low for B-Tree, total disk accesses for most of the operations are reduced significantly compared to balanced Binary Search Trees like AVL Tree, Red Black Tree,etc.

1.4 Properties to B-Tree

1) A B-Tree is defined by the term *minimum degree* ‘t’. The value of t depends upon disk.

2) Every node except root must contain at least t-1 keys. Root may contain minimum 1key.

3) All nodes (including root) may contain at most 2t – 1 key.

5) All keys of a node are sorted in increasing order. The child between two keys k1.

6) B-Tree grows and shrinks from root which is unlike Binary Search Tree.

CHAPTER 2

2.1 Algorithms used

==Algorithms==

{{confusing|reason=the discussion below uses "element", "value", "key", "separator", and "separation value" to mean essentially the same thing. The terms are not clearly defined. There are some subtle issues at the root and leaves

===Search===

Searching is similar to searching a binary search tree. Starting at the root, the tree is recursively traversed from top to bottom. At each level, the search reduces its field of view to the child pointer (Sub-Tree) whose range includes the search value. A Sub-Tree's range is defined by the values, or keys, contained in its parent node. These limiting values are also known as separation values.

Binary search is typically (but not necessarily) used within nodes to find the separation values and child tree of interest.

===Insertion===

[[Image: B-Tree insertion example.png|thumb|A B-Tree insertion example with each iteration. The nodes of this B-Tree have at most 3 children (Knuth order 3).]]

All insertions start at a leaf node. To insert a new element, search the tree to find the leaf node where the new element should be added. Insert the new element into that node with the following steps:

# If the node contains fewer than the maximum allowed number of elements, then there is room for the new element. Insert the new element in the node, keeping the node's elements ordered.

Otherwise the node is full, evenly split it into two nodes so:

A single median is chosen from among the leaf's elements and the new element.

Values less than the median are put in the new left node and values greater than the median are put in the new right node, with the median acting as a separation value.

The separation value is inserted in the node's parent, which may cause it to be split, and so on. If the node has no parent (i.e., the node was the root), create a new root above this node (increasing the height of the tree).

If the splitting goes all the way up to the root, it creates a new root with a single separator value and two children, which is why the lower bound on the size of internal nodes does not apply to the root. The maximum number of elements per node is "U"−1. When a node is split, one element moves to the parent, but one element is added. So, it must be possible to divide the maximum number "U"−1 of elements into two legal nodes. If this number is odd, then

"U"=2"L" and one of the new nodes contains ("U"−2)/2 = "L"−1 elements, and hence is a legal node, and the other contains one more element, and hence it is legal too. If "U"−1 is even, then "U"=2"L"−1, so there are 2"L"−2 elements in the node. Half of this number is

"L"−1, which is the minimum number of elements allowed per node.

An improved algorithm {{Citation needed|date=August 2015}} supports a single pass down the tree from the root to the node where the insertion will take place, splitting any full nodes encountered on the way. This prevents the need to recall the parent nodes into memory, which may be expensive if the nodes are on secondary storage. However, to use this improved algorithm, we must be able to send one element to the parent and split the remaining "U"−2 elements into two legal nodes, without adding a new element. This requires "U" = 2"L" rather than "U" = 2"L"−1, which accounts for why some textbooks impose this requirement in defining B-Trees.[5]

===Deletion===

There are two popular strategies for deletion from a B-Tree.

# Locate and delete the item, then restructure the tree to retain its invariants, '''OR'''
# Do a single pass down the tree, but before entering (visiting) a node, restructure the tree so that once the key to be deleted is encountered, it can be deleted without triggering the need for any further restructuring

The algorithm below uses the former strategy.

There are two special cases to consider when deleting an element:

# The element in an internal node is a separator for its child nodes
# Deleting an element may put its node under the minimum number of elements and children The procedures for these cases are in order below.

====Deletion from a leaf node====

# Search for the value to delete.

# If the value is in a leaf node, simply delete it from the node.

# If underflow happens, rebalance the tree as described in section "Rebalancing after deletion" below.

====Deletion from an internal node====

Each element in an internal node acts as a separation value for two Sub-Trees, therefore we need to find a replacement for separation. Note that the largest element in the left Sub-Tree is still less than the separator. Likewise, the smallest element in the right Sub-Tree is still greater than the separator. Both of those elements are in leaf nodes, and either one can be the new separator for the two Sub-Trees. Algorithmically described below:

# Choose a new separator (either the largest element in the left Sub-Tree or the smallest element in the right Sub-Tree), remove it from the leaf node it is in, and replace the element to be deleted with the new separator.

The previous step deleted an element (the new separator) from a leaf node. If that leaf node is now deficient (has fewer than the required number of nodes), then rebalance the tree starting from the leaf node.

2.2 Classification of Requirements

2.2 **User Requirements**

Operating System: Windows/Linux.

2.3 **Software and Hardware Requirements**

Programming Languages: Java

Chapter 3

IMPLEMENTATION

B-Tree is a self-balanced tree (i.e. all the leaf nodes have the same height level) data structure. However, unlike other trees such as binary tree, red-black and AVL tree whose nodes have only 2 children: left child node and right child node, B-Tree's nodes have more than 2 child nodes. So sometimes it is called M-way branching tree due to M number of children (M >= 2) that a node in B-Tree can have.[6]

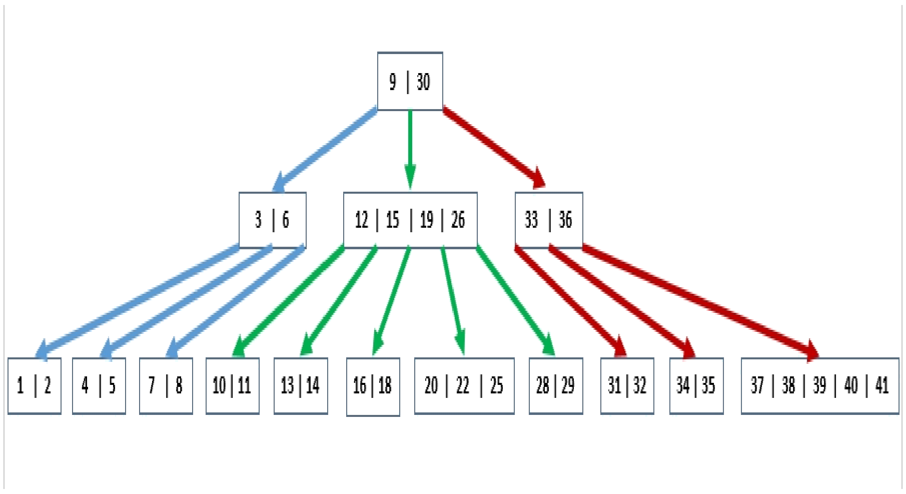


Fig 3.1 An example of a B-Tree

Root node, internal node and leaf node

Internal nodes are nodes that have children. Internal nodes lie above the bottom of the tree. In the figure 1, node [3 | 6], node [12 | 15 | 19| 26] and node [33 | 36] are internal nodes.

Leaf nodes are nodes that don't have children. They are the nodes at the bottom of the tree. In the figure 1, node [1 | 2], node [4 | 5], node [20 | 22 | 25] and node [31 | 32] are some of leaf nodes.

The root node of B-Tree is a special node. There is only one root node for B-Tree and it lies at the top of the tree. Depending on the number of items in B-Tree, the root node can be either an internal node or a leaf node. Node [9 | 30] is the root node of the B-Tree in Figure 3.1.

B-Tree node's properties

Each node can have a bunch of keys and a bunch of children (child nodes) where number of children can be either 0 or total number of its keys plus 1. Let consider node[x]. If node[x] is a leaf node then it won't have any children. If it is an internal node then its total number of children is n[x] + 1 where n[x] is the total number of its keys.

B-Tree's constraints:

Let ***t*** be the minimum degree of the B-Tree where ***t*** >= 2

Constraint #1: Every node other than the root node must have at least (**t** – 1) keys. It is the lower bound for the total number of keys in B-Tree's node.

Constraint #2: Every node including the root node must have at most (2***t*** – 1) keys. So we say the node is full if it has (2***t*** – 1) keys. It is the upper bound for the total number of keys in B-Tree's node.

Constraint #3: Every internal node (other than the root node) must have at least ***t*** children. Every internal node (including the root node) must have at most 2***t*** children.

Constraint #4: The keys in a node must be stored in ascending order. For example in Figure 1, node [12 | 15 | 19 | 26] has key 12 < key 15 < key 19 < key 26

Constraint #5: All the keys of child nodes that are on the left side of a key must be smaller than that key. In Figure 1, child nodes that are on the left side of key 30 are node [12 | 15 | 19 | 26], node [10 | 11], node [13 | 14] , node [16 | 18], node [20 | 22 | 25] and node [28 | 29] have their keys smaller than 30.

Constraint #6: All the keys of child nodes that are on the right side of a key must be larger than that key. For example in Figure 1, child nodes that are on the right side of key 9 are node [12 | 15 | 19 | 26], node [10 | 11], node [13 | 14] , node [16 | 18], node [20 | 22 | 25] and node [28 | 29] have their keys larger than 9.

With *constraint #4* and *constraint#5*, generally all nodes on the left side of a key must have keys smaller than it.

With *constraint #4* and *constraint#6*, generally all nodes on the right side of a key must have keys larger than it.

In Figure 3.1, the B-Tree's minimum degree is 3. So, its lower bound is 2 and its upper bound is 5.

If you pay close attention to the example of B-Tree in Figure 1, you'll notice that a key in any node is actually a range separator of all the keys in the nodes at the lower levels on that key's left side and right side.

Let look at node [9 | 30], the keys of lower nodes on the left side of key 9 (keys in the nodes that are linked by blue arrows) are smaller than 9. The keys of lower nodes on the right side of key 9 (keys in the nodes that are linked by green arrows) are larger than 9. The keys of lower nodes on the left side of key 30 (keys in the nodes that are linked by green arrows) are smaller than 30. The keys of lower nodes on the right side of key 30 (keys in the nodes that are linked by red arrows) are larger than 30. This pattern of B-Tree makes the key searching is similar to binary-tree's key searching.

As long as the B-Tree operations don't violate the above constraints, it is automatically self-balanced. In other words, the constraints are devised as such to keep its balance property in check.

The minimum degree (**t**) is inversely proportional to B-Tree height (**h**). Increasement

for **t** will decrease **h**. However larger **t** means more time to spend in nodes for searching keys and traversing child nodes.

The concepts and pseudo-code for B-Tree operations:

In order to understand how basic B-Tree operations like key searching, insertion and deletion are implemented, some key concepts I would like to introduce before we go to the full implementation details.

Right and left sibling

Right and left sibling of a node are the nodes on its right and left side at the same level.

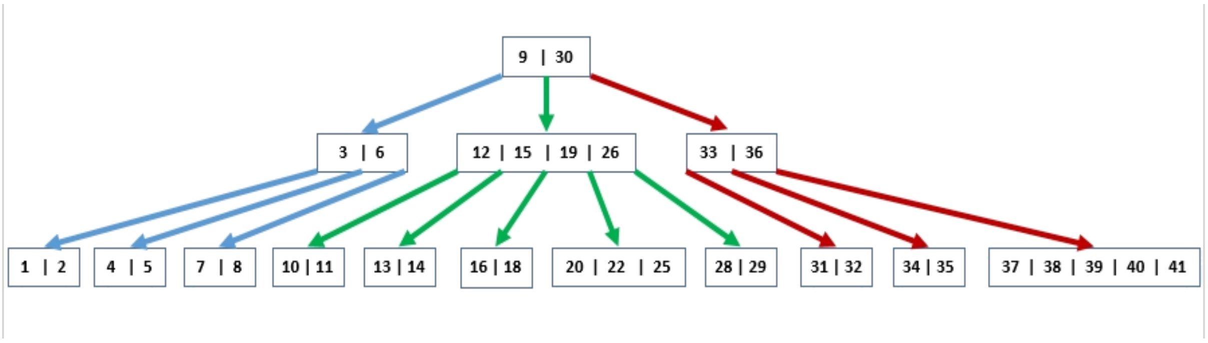


Fig 3.2 Node siblings

In Figure 2, the left sibling of node [18 | 22] is node [3 | 12] and its right sibling is node [33 | 36]. Node [3 | 2] doesn't have left sibling and its right sibling is node [18 | 22]. The left sibling of node [33 | 36] is node [18 | 22] and it doesn't have right sibling.

Random example:

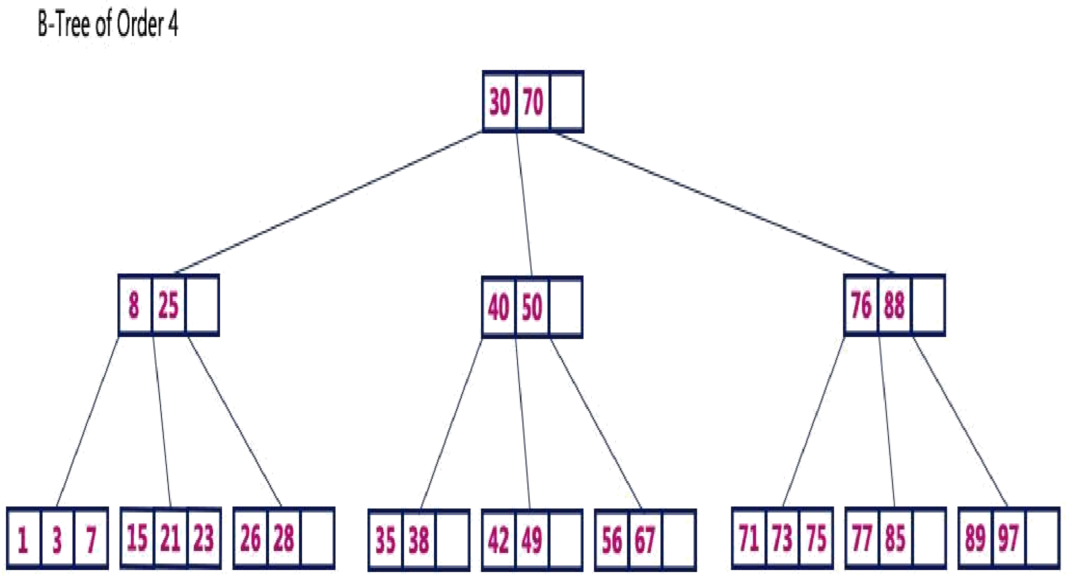


Fig 3.3 Random example

CHAPTER 4

RESULT AND ANALYSIS

4.1 B-Tree Analysis:

- Let and n/2 and e=[n-1/2]
- Suppose that a B-Tree consists of levels, where the root is level 0.
- Then level i, where, must contain at least di-1 nodes, otherwise some node at level i-1 or less would contain fewer than d children, which is not possible, or the root would have fewer than one child, which is also not possible.
- The total number of leaf nodes is therefore at least dm-2.
- The total number of search key values k is therefore at least edm-2, i.e.,
- Dividing by e and taking the log of both sides gives • Thus, the height of the tree m is logarithmic in k, which is the number of search key values in the tree.
- Although this isn't constant time, note that d is typically quite large and, consequently, m is quite small.

4.2 B-Tree generation time vs Data graph :

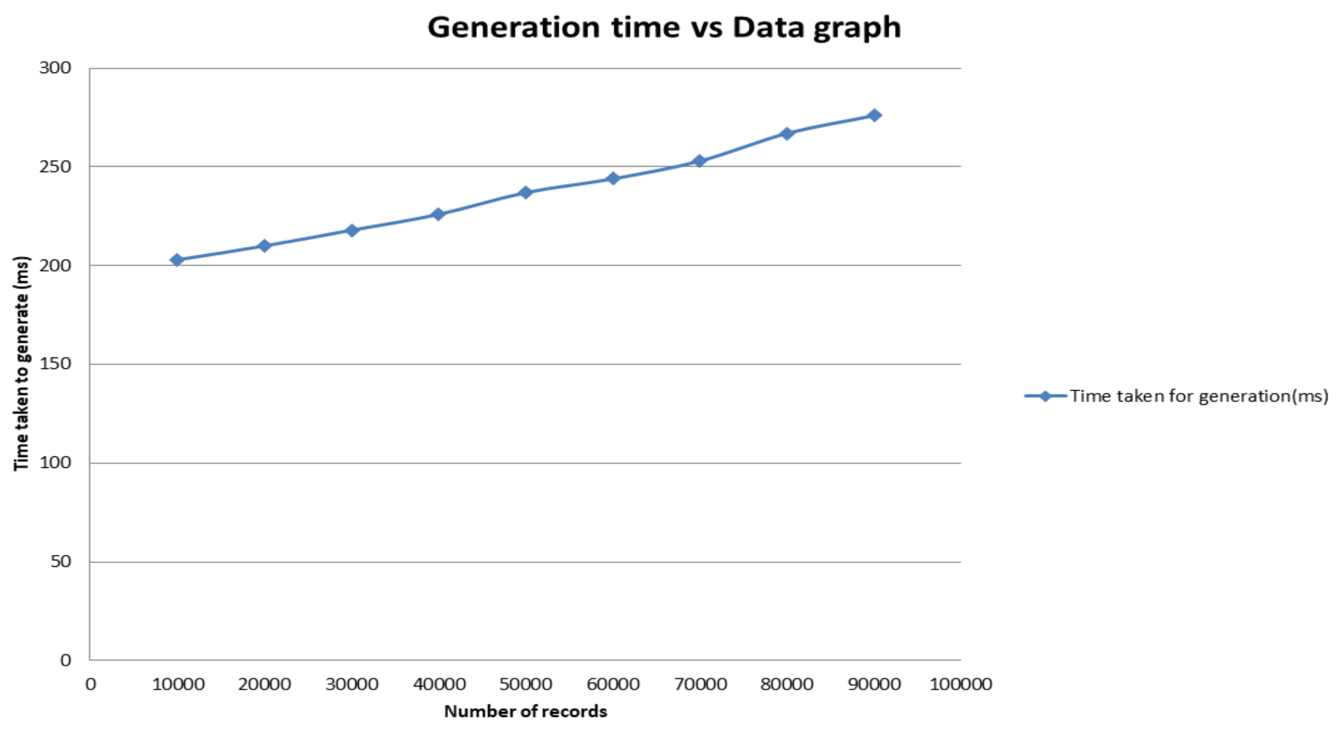


Fig 4.1 Generation of B-Tree

- The above graph represents the time complexity of B-Tree for given data set with records ranging from 10,000-1,00,000 with time 1 to 5.8 mili seconds.

4.3 B-Tree insertion time vs Data graph :

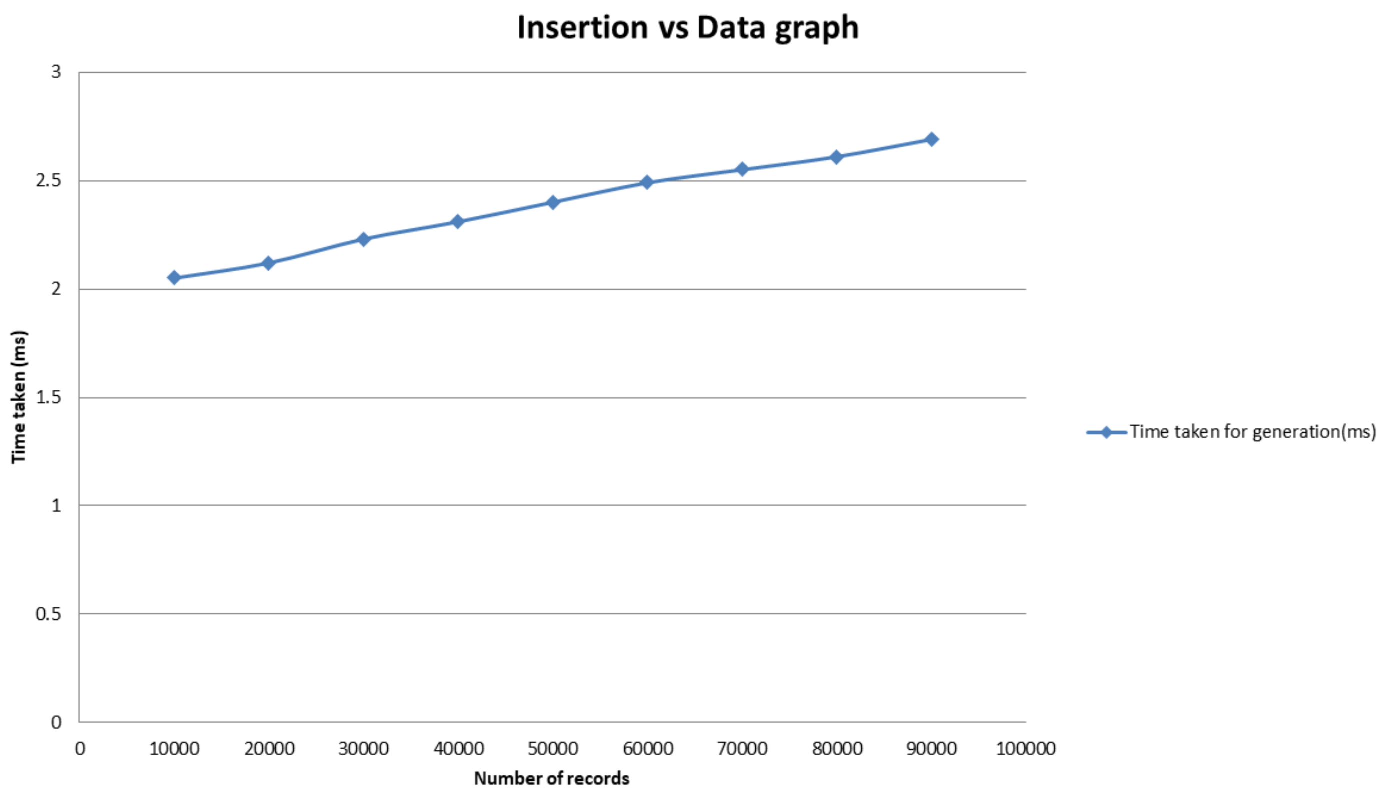


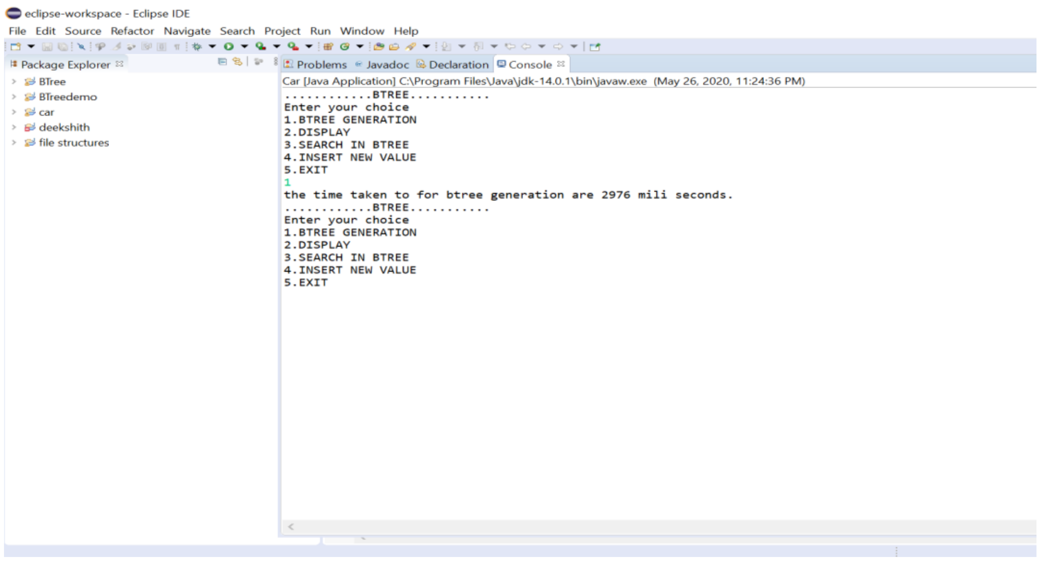
Fig 4.2 Insertion in B-Tree

- The above graph represents the complexity of insertion in B-Tree for given data set with records ranging from 10,000-1,00,000 with time10,000-0.65ms, 20,000-.87ms ,...... 1,00,000-2.9ms.

CHAPTER 5

Screenshots of output:

1. B-Tree generation

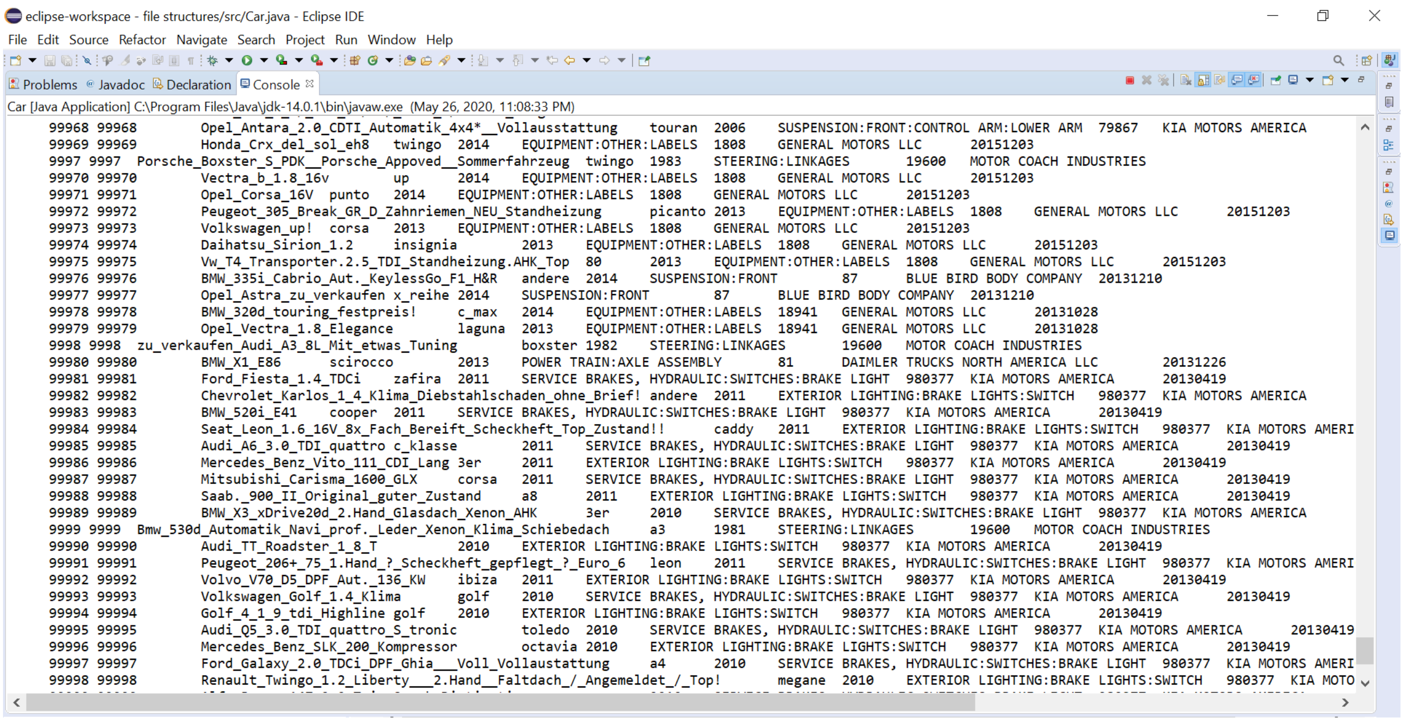

eclipse-workspace - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
BTree
BTreedemo
car
deekshith
file structures
Problems
Javadoc
Declaration
Console

```
Car [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe  (May 26, 2020, 11:24:36 PM)
............BTREE...........
Enter your choice
1.BTREE GENERATION
2.DISPLAY
3.SEARCH IN BTREE
4.INSERT NEW VALUE
5.EXIT
1
the time taken to for btree generation are 2976 mili seconds.
............BTREE...........
Enter your choice
1.BTREE GENERATION
2.DISPLAY
3.SEARCH IN BTREE
4.INSERT NEW VALUE
5.EXIT
```

Fig 5.1 Generation of B-Tree:

- In the above figure user choose option 1 so B-Tree is generated and the time for generation is displayed in milli seconds.

2. Display of B-Tree


eclipse-workspace - file structures/src/Car.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console

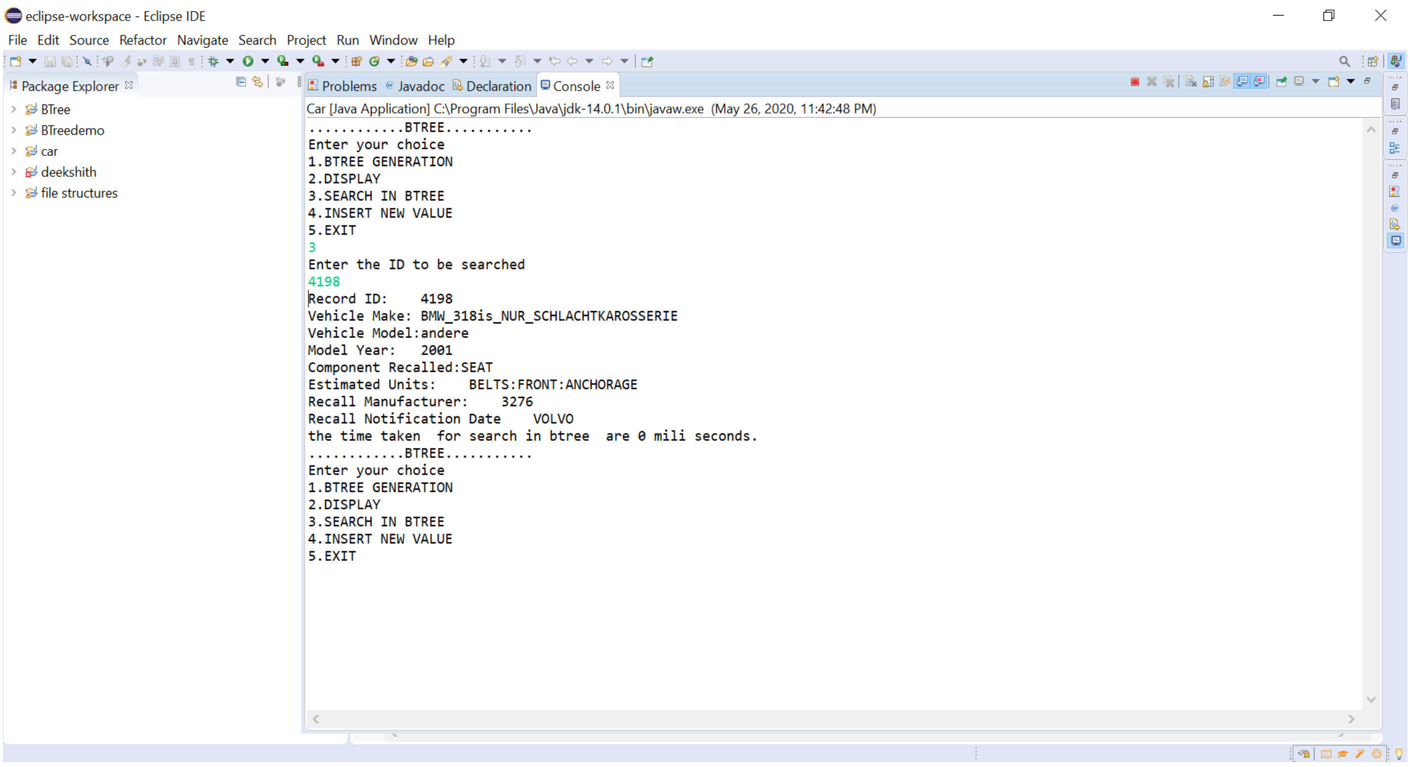
```
Car [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe  (May 26, 2020, 11:08:33 PM)
      99968 99968        Opel_Antara_2.0_CDTI_Automatik_4x4*__Vollausstattung    touran  2006    SUSPENSION:FRONT:CONTROL ARM:LOWER ARM  79867   KIA MOTORS AMERICA
      99969 99969        Honda_Crx_del_sol_eh8   twingo  2014    EQUIPMENT:OTHER:LABELS  1808    GENERAL MOTORS LLC      20151203
      9997 9997  Porsche_Boxster_S_PDK__Porsche_Appoved__Sommerfahrzeug  twingo  1983    STEERING:LINKAGES       19600   MOTOR COACH INDUSTRIES
      99970 99970        Vectra_b_1.8_16v        up      2014    EQUIPMENT:OTHER:LABELS  1808    GENERAL MOTORS LLC      20151203
      99971 99971        Opel_Corsa_16V  punto   2014    EQUIPMENT:OTHER:LABELS  1808    GENERAL MOTORS LLC      20151203
      99972 99972        Peugeot_305_Break_GR_D_Zahnriemen_NEU_Standheizung      picanto 2013    EQUIPMENT:OTHER:LABELS  1808    GENERAL MOTORS LLC      20151203
      99973 99973        Volkswagen_up!  corsa   2013    EQUIPMENT:OTHER:LABELS  1808    GENERAL MOTORS LLC      20151203
      99974 99974        Daihatsu_Sirion_1.2     insignia        2013    EQUIPMENT:OTHER:LABELS  1808    GENERAL MOTORS LLC      20151203
      99975 99975        Vw_T4_Transporter.2.5_TDI_Standheizung.AHK_Top  80      2013    EQUIPMENT:OTHER:LABELS  1808    GENERAL MOTORS LLC      20151203
      99976 99976        BMW_335i_Cabrio_Aut._KeylessGo_F1_H&R   andere  2014    SUSPENSION:FRONT        87      BLUE BIRD BODY COMPANY  20131210
      99977 99977        Opel_Astra_zu_verkaufen x_reihe 2014    SUSPENSION:FRONT        87      BLUE BIRD BODY COMPANY  20131210
      99978 99978        BMW_320d_touring_festpreis!     c_max   2014    EQUIPMENT:OTHER:LABELS  18941   GENERAL MOTORS LLC      20131028
      99979 99979        Opel_Vectra_1.8_Elegance        laguna  2013    EQUIPMENT:OTHER:LABELS  18941   GENERAL MOTORS LLC      20131028
      9998 9998  zu_verkaufen_Audi_A3_8L_Mit_etwas_Tuning        boxster 1982    STEERING:LINKAGES       19600   MOTOR COACH INDUSTRIES
      99980 99980        BMW_X1_E86      scirocco        2013    POWER TRAIN:AXLE ASSEMBLY       81      DAIMLER TRUCKS NORTH AMERICA LLC        20131226
      99981 99981        Ford_Fiesta_1.4_TDCi    zafira  2011    SERVICE BRAKES, HYDRAULIC:SWITCHES:BRAKE LIGHT  980377  KIA MOTORS AMERICA      20130419
      99982 99982        Chevrolet_Karlos_1_4_Klima_Diebstahlschaden_ohne_Brief! andere  2011    EXTERIOR LIGHTING:BRAKE LIGHTS:SWITCH   980377  KIA MOTORS AMERICA
      99983 99983        BMW_520i_E41    cooper  2011    SERVICE BRAKES, HYDRAULIC:SWITCHES:BRAKE LIGHT  980377  KIA MOTORS AMERICA      20130419
      99984 99984        Seat_Leon_1.6_16V_8x_Fach_Bereift_Scheckheft_Top_Zustand!!      caddy   2011    EXTERIOR LIGHTING:BRAKE LIGHTS:SWITCH   980377  KIA MOTORS AMERI
      99985 99985        Audi_A6_3.0_TDI_quattro c_klasse        2011    SERVICE BRAKES, HYDRAULIC:SWITCHES:BRAKE LIGHT  980377  KIA MOTORS AMERICA      20130419
      99986 99986        Mercedes_Benz_Vito_111_CDI_Lang 3er     2011    EXTERIOR LIGHTING:BRAKE LIGHTS:SWITCH   980377  KIA MOTORS AMERICA      20130419
      99987 99987        Mitsubishi_Carisma_1600_GLX     corsa   2011    SERVICE BRAKES, HYDRAULIC:SWITCHES:BRAKE LIGHT  980377  KIA MOTORS AMERICA      20130419
      99988 99988        Saab._900_II_Original_guter_Zustand     a8      2011    EXTERIOR LIGHTING:BRAKE LIGHTS:SWITCH   980377  KIA MOTORS AMERICA      20130419
      99989 99989        BMW_X3_xDrive20d_2.Hand_Glasdach_Xenon_AHK      3er     2010    SERVICE BRAKES, HYDRAULIC:SWITCHES:BRAKE LIGHT  980377  KIA MOTORS AMERICA
      9999 9999  Bmw_530d_Automatik_Navi_prof._Leder_Xenon_Klima_Schiebedach     a3      1981    STEERING:LINKAGES       19600   MOTOR COACH INDUSTRIES
      99990 99990        Audi_TT_Roadster_1_8_T          2010    EXTERIOR LIGHTING:BRAKE LIGHTS:SWITCH   980377  KIA MOTORS AMERICA      20130419
      99991 99991        Peugeot_206+_75_1.Hand_?_Scheckheft_gepflegt_?_Euro_6   leon    2011    SERVICE BRAKES, HYDRAULIC:SWITCHES:BRAKE LIGHT  980377  KIA MOTORS AMERI
      99992 99992        Volvo_V70_D5_DPF_Aut._136_KW    ibiza   2011    EXTERIOR LIGHTING:BRAKE LIGHTS:SWITCH   980377  KIA MOTORS AMERICA      20130419
      99993 99993        Volkswagen_Golf_1.4_Klima       golf    2010    SERVICE BRAKES, HYDRAULIC:SWITCHES:BRAKE LIGHT  980377  KIA MOTORS AMERICA      20130419
      99994 99994        Golf_4_1_9_tdi_Highline golf    2010    EXTERIOR LIGHTING:BRAKE LIGHTS:SWITCH   980377  KIA MOTORS AMERICA      20130419
      99995 99995        Audi_Q5_3.0_TDI_quattro_S_tronic        toledo  2010    SERVICE BRAKES, HYDRAULIC:SWITCHES:BRAKE LIGHT  980377  KIA MOTORS AMERICA      20130419
      99996 99996        Mercedes_Benz_SLK_200_Kompressor        octavia 2010    EXTERIOR LIGHTING:BRAKE LIGHTS:SWITCH   980377  KIA MOTORS AMERICA      20130419
      99997 99997        Ford_Galaxy_2.0_TDCi_DPF_Ghia___Voll_Vollaustattung     a4      2010    SERVICE BRAKES, HYDRAULIC:SWITCHES:BRAKE LIGHT  980377  KIA MOTORS AMERI
      99998 99998        Renault_Twingo_1.2_Liberty___2.Hand__Faltdach_/_Angemeldet_/_Top!       megane  2010    EXTERIOR LIGHTING:BRAKE LIGHTS:SWITCH   980377  KIA MOTO
```

Fig 5.2 Display of B-Tree

- In the above figure the B-Tree written into a file is displayed. T he line number indicates the level of B-Tree. Each number is a node.

3. Search an element in B-Tree

When the option 3 is chosen,



eclipse-workspace - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
BTree
BTreedemo
car
deekshith
file structures
Problems
Javadoc
Declaration
Console

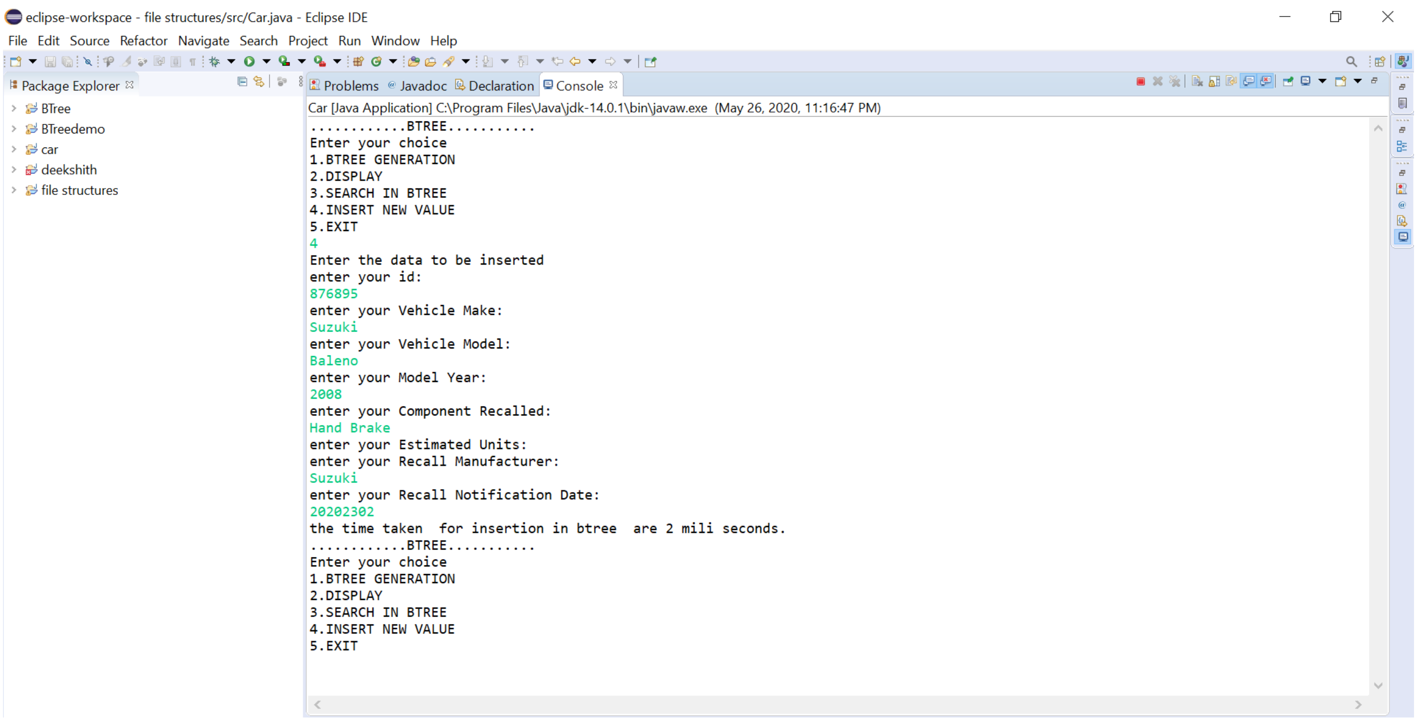
```
Car [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe  (May 26, 2020, 11:42:48 PM)
............BTREE...........
Enter your choice
1.BTREE GENERATION
2.DISPLAY
3.SEARCH IN BTREE
4.INSERT NEW VALUE
5.EXIT
3
Enter the ID to be searched
4198
Record ID:    4198
Vehicle Make: BMW_318is_NUR_SCHLACHTKAROSSERIE
Vehicle Model:andere
Model Year:   2001
Component Recalled:SEAT
Estimated Units:    BELTS:FRONT:ANCHORAGE
Recall Manufacturer:    3276
Recall Notification Date    VOLVO
the time taken  for search in btree  are 0 mili seconds.
............BTREE...........
Enter your choice
1.BTREE GENERATION
2.DISPLAY
3.SEARCH IN BTREE
4.INSERT NEW VALUE
5.EXIT
```

Fig 5.3 Search an element in B-Tree

- In the above figure we are giving id“4198”.
- Then matching car ID data is displayed.

4. Inserting an element into B-Tree

When option 4 is chosen,


eclipse-workspace - file structures/src/Car.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
BTree
BTreedemo
car
deekshith
file structures
Problems
Javadoc
Declaration
Console

Car [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe (May 26, 2020, 11:16:47 PM)

```
............BTREE...........
Enter your choice
1.BTREE GENERATION
2.DISPLAY
3.SEARCH IN BTREE
4.INSERT NEW VALUE
5.EXIT
4
Enter the data to be inserted
enter your id:
876895
enter your Vehicle Make:
Suzuki
enter your Vehicle Model:
Baleno
enter your Model Year:
2008
enter your Component Recalled:
Hand Brake
enter your Estimated Units:
enter your Recall Manufacturer:
Suzuki
enter your Recall Notification Date:
20202302
the time taken  for insertion in btree  are 2 mili seconds.
............BTREE...........
Enter your choice
1.BTREE GENERATION
2.DISPLAY
3.SEARCH IN BTREE
4.INSERT NEW VALUE
5.EXIT
```

Fig 5.4 Inserting an element into B-Tree

- In the above figure the we are inserting a data given from the user into B-Tree.
- And the new data has been inserted to the data set.

CONCLUSION:

We have successfully implemented B-TREE which helps us in administrating the data used for managing the tasks performed.

View tables are used to display all the components at once so that user can see all the components of a Particular type at once. One can just select the component and modify and remove the component.

We have successfully used various functionalities of JAVA and created the File structures. Features:

1. Clean separation of various components to facilitate easy modification and revision.
2. All the data is maintained in a separate file to facilitate easy modification
3. All the data required for different operations is kept in a separate file.
4. Quick and easy saving and loading of database file.

REFERENCE:

[1] en.wikipedia.org/wiki/B-Tree

[2] www.geeksforgeeks.org/B-Tree-set-1-introduction-2/

[3] btechsmartclass.com/DS/U5_T3.html

[4] lcm.csa.iisc.ernet.in/dsa/node122.html

[5] www.cpp.edu/~ftang/courses/CS241/notes/B-Tree.htm

[6] searchsqlserver.techtarget.com/definition/B-Tree