VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belagavi, Karnataka 590018

A MINI-PROJECT REPORT
on

IMPLEMENTATION OF B+ TREE ON PHARMACY OBJECTS

FILE STRUCTURES LABORATORY (15ISL68)

Submitted in partial fulfillment of the requirement
for the award of the degree of

Bachelor of Engineering
in

Information Science & Engineering
by

Sarthak Sureka (1BG15IS044)

Vidyaya Amrutham Ashnuthe

B.N.M. Institute of Technology

(Approved by AICTE, Affiliated to VTU, ISO 9001:2008 Certified
and Accredited as grade A Institution by NAAC)
Post box no. 7087, 27th cross, 12th Main,
Banashankari II Stage, Bengaluru- 560070, INDIA
Ph: 91-80- 26711780/81/82 Email: principal@bnmit.in, bnmitprincipal@gmail.com, www. bnmit.org

Department of Information Science and Engineering
2017 – 2018

B.N.M. Institute of Technology

(Approved by AICTE, Affiliated to VTU, ISO 9001:2008 Certified and Accredited as grade A Institution by NAAC)
Post box no. 7087, 27th cross, 12th Main,
Banashankari II Stage, Bengaluru- 560070, INDIA
Ph: 91-80- 26711780/81/82 Email: principal@bnmit.in, bnmitprincipal@gmail.com, www. bnmit.org

Department of Information Science & Engineering

Vidyaya Amrutham Ashnuthe

CERTIFICATE

Certified that the Mini-project entitled **Implementation of B+ tree on Pharmacy objects** is carried out by Mr. **Sarthak Sureka** USN **1BG15IS044** the bonafide student of **B.N.M Institute of Technology** in partial fulfillment for the award of **Bachelor of Engineering** in **Information Science & Engineering** of the **Visvesvaraya Technological University**, Belagavi during the year 2017-2018. It is certified that all corrections / suggestions indicated for Internal Assessment have been incorporated in the report deposited in the department library. The mini-project report has been approved as it satisfies the academic requirements in respect of mini-project prescribed for the said Degree.

Prof. Laxmi V.	Dr. Surabhi Narayan
Asst. Prof., Dept. of ISE	**Prof & Head, Dept. of ISE**
BNMIT	BNMIT

Name of the Examiners	**Signature with date**
1.	
2.	

Table of Contents

List of Figures

CHAPTER 1

INTRODUCTION

File structures is an organization of data in Secondary Storage Device in such a way that it minimizes the access time and the storage space. It is a combination of representation for data in files and of operations for accessing the data.

Early in the computing history, secondary storage was in the form of magnetic tape and punched cards. Storage was cheap but access was limited to sequential. In 1956, IBM introduced the RAMAC magnetic disk device. Data could be accessed directly instead of sequentially. This was the dawn of the study of file structures. Advances in OS gave rise to more research on operating systems. The next analogy that was come up was the Direct Access which is the analogy to access to position in array. Indexes were invented and list of keys and pointers were stored in small files. This allowed direct access to a large primary file. But as the file grows the same problem arise as with the primary memory. Tree structures emerged for main memory in 1960's. This involved balanced and self-adjusting Binary Search trees (BST) eg: AVL trees (1963). In 1979, a tree structure suitable for files was invented: B trees and B+ trees good for accessing millions of records with three or four disk accesses. Theory on Hashing tables were developed over 60's and 70's goof for those files that do not change much over time. Expandable and dynamic Hashing were invented in late 70's and 80's which provided for one or two disk accesses even if file grow dramatically.

Problem Statement :- Develop a mini project to implement B+ tree for a file of pharmacy objects. Implement insert(), search() and traversal().

A B+ tree is an N-ary tree with a variable but often large number of children per node. A B+ tree consists of a root, internal nodes and leaves. The root may be either a leaf or a node with two or more children. A B+ tree can be viewed as a B-tree in which each node contains only keys (not key–value pairs), and to which an additional level is added at the bottom with linked leaves.

The primary value of a B+ tree is in storing data for efficient retrieval in a block-oriented storage context — in particular, file systems. This is primarily because unlike binary search trees, B+ trees have very high fan out (number of pointers to child nodes in a node, typically on the order of 100 or more), which reduces the number of I/O operations required to find an element in the tree.

The leaves (the bottom-most index blocks) of the B+ tree are often linked to one another in a linked list; this makes range queries or an (ordered) iteration through the blocks simpler and more efficient (though the aforementioned upper bound can be achieved even without this addition). This does not substantially increase space consumption or maintenance on the tree. This illustrates one of the significant advantages of a B+ tree over a B-tree; in a B-tree, since not all keys are present in the leaves, such an ordered linked list cannot be constructed. A B+ tree is thus particularly useful as a database system index, where the data typically resides on disk, as it allows the B+ tree to actually provide an efficient structure for housing the data itself.

If a storage system has a block size of B bytes, and the keys to be stored have a size of k, arguably the most efficient B+ tree is one where b = B/k -1. Although theoretically the one-off is unnecessary, in practice there is often a little extra space taken

up by the index blocks (for example, the linked list references in the leaf blocks). Having an index block which is slightly larger than the storage system's actual block represents a significant performance decrease; therefore erring on the side of caution is preferable. If nodes of the B+ tree are organized as arrays of elements, then it may take a considerable time to insert or delete an element as half of the array will need to be shifted on average. To overcome this problem, elements inside a node can be organized in a binary tree or a B+ tree instead of an array.

B+ trees can also be used for data stored in RAM. In this case a reasonable choice for block size would be the size of processor's cache line.

Space efficiency of B+ trees can be improved by using some compression techniques. One possibility is to use delta encoding to compress keys stored into each block. For internal blocks, space saving can be achieved by either compressing keys or pointers. For string keys, space can be saved by using the following technique: Normally the *i*-th entry of an internal block contains the first key of block (i+1). Instead of storing the full key, we could store the shortest prefix of the first key of block (i+1) that is strictly greater (in lexicographic order) than last key of block *i*. There is also a simple way to compress pointers: if we suppose that some consecutive blocks i, i+1,…..i+k are stored contiguously, then it will suffice to store only a pointer to the first block and the count of consecutive blocks.

All the above compression techniques have some drawbacks. First, a full block must be decompressed to extract a single element. One technique to overcome this problem is to divide each block into sub-blocks and compress them separately. In this case searching or inserting an element will only need to decompress or compress a sub-block instead of a full block. Another drawback of compression techniques is that the number of stored elements may vary considerably from a block to another depending on how well the elements are compressed inside each block.

1.1 Requirements Analysis

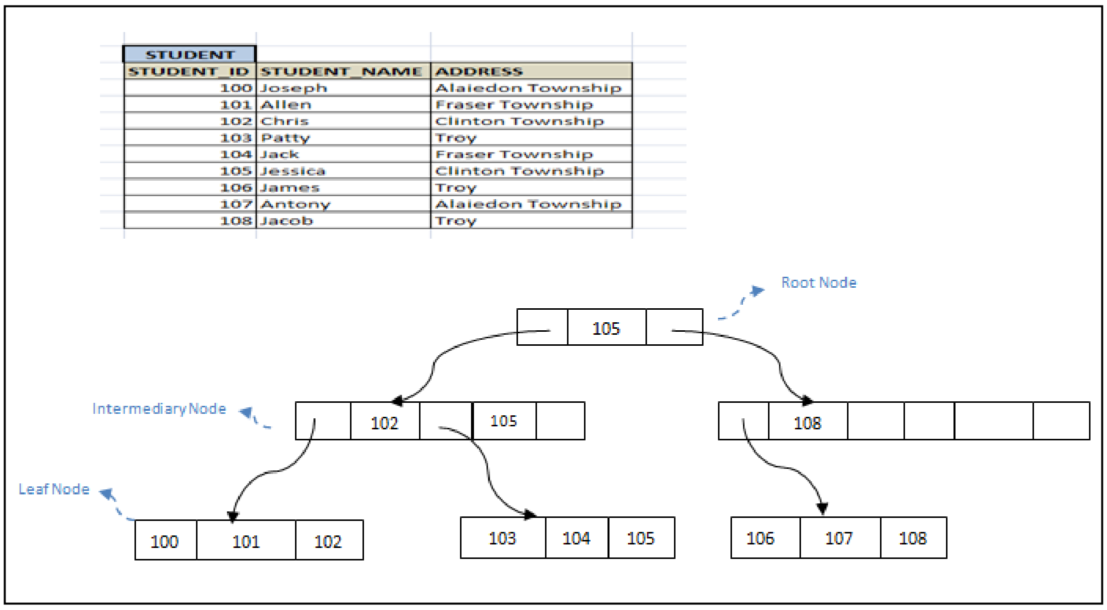
The project aims at developing a mini project that allows the end user to store information about pharmacy objects that it has linkup with, in a file with all the necessary details along with it. It helps the end user to insert a pharmacy detail and with a unique id that can be used to access the information of that pharmacy, it also helps the end user to search the information of a particular pharmacy with the help of the unique id that is assigned to it and at last it also allows the end user to traverse through all the records that he/she has stored in the file with the help of B+ tree. The mini project is programmed with the help of the technique called B+ tree that provides an easier way to insert and traverse through the information stored in the file. This mini project will help the end user to maintain all the information in a proper and sorted manner.

CHAPTER 2

DESIGN

B+ tree is a (key, value) storage method in a tree like structure. B+ tree has one root, any number of intermediary nodes (usually one) and a leaf node. Here all leaf nodes will have the actual records stored. Intermediary nodes will have only pointers to the leaf nodes; it not has any data. Any node will have only two leaves. This is the basic of any B+ tree.

Consider the STUDENT table below. This can be stored in B+ tree structure as shown below. We can observe here that it divides the records into two and splits into left node and right node. Left node will have all the values less than or equal to root node and the right node will have values greater than root node. The intermediary nodes at level 2 will have only the pointers to the leaf nodes. The values shown in the intermediary nodes are only the pointers to next level. All the leaf nodes will have the actual records in a sorted order.



STUDENT

STUDENT_ID	STUDENT_NAME	ADDRESS
100	Joseph	Alaiedon Township
101	Allen	Fraser Township
102	Chris	Clinton Township
103	Patty	Troy
104	Jack	Fraser Township
105	Jessica	Clinton Township
106	James	Troy
107	Antony	Alaiedon Township
108	Jacob	Troy

Figure 2.1 Example of B+ Tree

If we have to search for any record, they are all found at leaf node. Hence searching any record will take same time because of equidistance of the leaf nodes. Also they are all sorted. Hence searching a record is like a sequential search and does not take much time.

2.1 Data Flow Diagram

There are different notations to draw data flow diagrams, defining different visual representations for processes, data stores, data flow, and external entities that is shown in Figure 2.2.

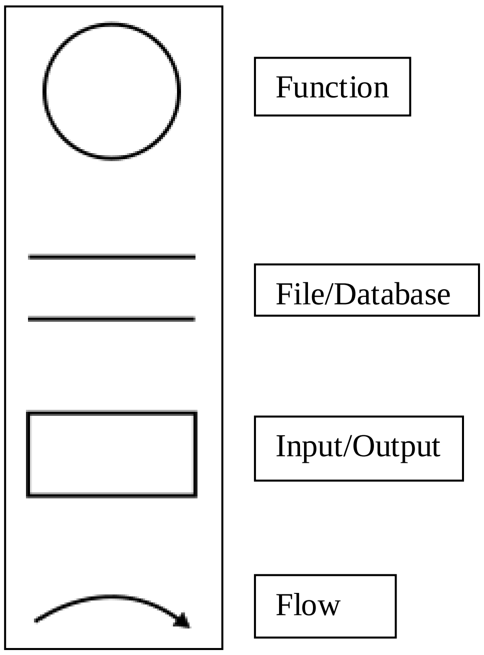


Figure 2.2 – Data Flow Diagram Notations

The Figure 2.3 shows the flow of the mini project that the user will experience. Firstly, the user has to validate his authenticity by entering the username and password. If the given data is correct the user is transferred to the system menu, where he gets a option to select from different operation that the user wants to perform. If the user selects insert operation, then the user needs to enter all the details that is asked in that form and then click save to save the details of the pharmacy in the file. After completing the operation press the back button to return to the system menu.

Similarly, if the user selects the search operation, then the user needs to enter the pharmacy id for which he wants to retrieve the details and then press search button which will search the file for that pharmacy id and display the details if that pharmacy id exist or else it will display an error message.

Else if the user wants to review all the information available in the file, then the user selects the display operation, which displays all the records that is saved in the file. Lastly, if the user selects the traversal option then the pharmacy id stored in the file are displayed in a tree format which represents a B+ tree.

The data flow diagram for pharmacy objects is represented in Figure 2.2.

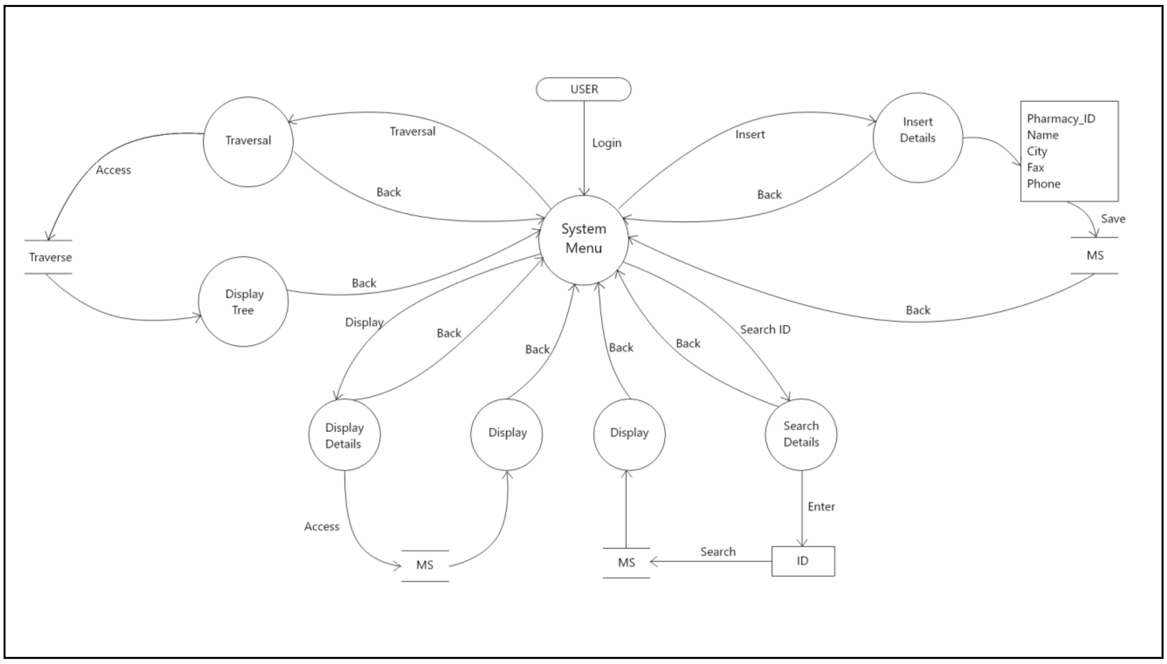


Figure 2.3 – Data Flow Diagram for Pharmacy Objects

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 Hardware and Software requirements

Hardware Requirements :-

- Processor :- Intel Core i5-7400 or Core i7-7700k clocked CPU @3.40GHz
- RAM :- 16GB DDR3
- Hard Disk :- 1TB at 7200 rpm
- GPU :- Nvidia GTX 1050 or Intel 640

Software Requirements :-

- Operating System :- Windows 10 pro 64 bit version 1803
- Netbeans IDE v8.1 (Complete Package)
- JDK version 7.1

3.2 Tools

NETBEANS IDE

Netbeans is a software development platform written in Java. The Netbeans Platform allows application to be developed from a set of modular software components called modules. Applications based on the Netbeans Platform, including the Netbeans integrated development environment (IDE), can be extended by third party developers.

The Netbeans IDE is primarily intended for development in Java, but also supports other languages, in particular PHP, C/C++ and HTML5.

Netbeans is cross-platform and runs on Microsoft Windows, mac OS, Linux, Solaris and other platforms supporting a compatible JVM.

The editor supports many languages from Java, C/C++, XML and HTML, to PHP, Groovy, Javadoc, JavaScript and JSP. Because the editor is extensible, you can plug in support for many other languages.

A new version was released 8.2 on October 3, 2016.NetBeans IDE is the official IDE for Java 8. With its editors, code analyzers, and converters, you can quickly and smoothly upgrade your applications to use new Java 8 language constructs, such as lambdas, functional operations, and method references.

Netbeans IDE is an open-source integrated development environment. Netbeans IDE supports development of all Java application types (JavaSE (including JavaFX), Java

ME, web, EJB and mobile applications) out of the box. Among other features are an Ant-based project system, Maven support, refactoring, and version control (supporting CVS, Subversion, Git, Mercurial and Clearcase).

JAVA DEVELOPMENT KIT (JDK)

The Java Development Kit (JDK) is an implementation of either one of the Java Platform, Standard Edition, Java Platform, Enterprise Edition, or Java Platform, Micro Edition platforms released by Oracle Corporation in the form of a binary product aimed at Java developers on Solaris, Linux, mac OS or Windows. The JDK includes a private JVM and a few other resources to finish the development of a Java Application. Since the introduction of the Java platform, it has been by far the most widely used Software Development Kit (SDK). On 17 November 2006, Sun announced that they would release it under the GNU General Public License (GPL), thus making it free software. This happened in large part on 8 May 2007, when Sun contributed the source code to the Open JDK.

CHAPTER 4

IMPLEMENTATION

B+ Tree is an advanced method of ISAM file organization. It uses the same concept of key-index, but in a tree like structure. B+ tree is similar to binary search tree, but it can have more than two leaf nodes. It stores all the records only at the leaf node. Intermediary nodes will have pointers to the leaf nodes. They do not contain any data/records. The different types of operations supported by B+ tree are :- insert, search, delete and traversal.

4.1 Description of Modules

The different operations supported by B+ tree and used to implement the pharmacy objects are listed below :-

1) Insert
2) Search
3) Display
4) Traversal

Insert module is used to insert details of the pharmacy to the file created by the user. This module inserts the information in an organised manner that is pre-defined and helps in proper management of all the information.

Search module is used to search for a specific pharmacy details that the user wish to review. This module searches the file with the help of the pharmacy id entered by the user and display's the information of the corresponding pharmacy.

Display module is used to review all the information that is stored in a file at once. This module can be used to see all pharmacy details that is stored in a file.

Traversal module is used to display the id of all the pharmacy in a tree format. This type of representation uses a technique called the B+ tree for displaying the pharmacy id.

4.2 Algorithms of Modules

The function used in the algorithms for implementation are as follows :-

A) FindLeaf :- This function is used to find the leaf node where we can insert a new node or it can be used to search for a particular node. The algorithm for the function is shown below :

```
Step 1: Start
Step 2: Initialise two node objects, ptr and prevptr, and declare an integer
        variable i
```

```
Step 3: Till ptr equals NULL, increment the value of i to find the size of the tree,
        after that point the pointer to the next node on the tree
Step 4: Return prevptr
Step 5: End
```

B) Promote :- This function traverses the tree to find the correct position where the node can be inserted. The algorithm for the function is shown below:

```
Step 1: Start
Step 2: If n equals NULL, then return to the previous node.
Step 3: If n.size less than 4, then the insertIntoNode function is called, which insert's
        the value into the concerned node and then return to the previous node.
Step 4: If n equals root, then create a new root node and make it point to the parent
        node.
Step 5: Declare a new node that splits the parent node with the help of the split
        function that takes the node to be splitted as argument.
Step 6: If key is less than n.a[0], then declare a object of node class and assign it the
        value of newptr else assign the value of n.
Step 7: Call the InsertIntoNode function with the arguments, n initialise to t, key to
        key and address to address to insert a node into the tree.
Step 8: Call promote function for the arguments, n initialised to parent node, key to
        the number of values stored in n and address to n.
Step 9: Repeat the process from Step 6 to execute promote function for the arguments,
        n initialised to newptr parent node, key to number of values stored in newptr
        and address to newptr.
Step 10: End
```

C) InsertIntoNode :- This function is used to insert the key into the node or it can also be used to update the value of a node. The algorithm for the function is shown below:

```
Step 1: Start
Step 2: Declare an integer variable i
Step 3: If n equals NULL, then return the value of node.
Step 4: For all the values in the tree, compare it with the key and return if it is equal to
        the tree.
Step 5: Initialise i to the end of the node
Step 6: For value greater than key, assign the values to the next node in the tree and
        decrement i by 1.
Step 7: Increment i by 1.
Step 8: Assign the key to the node
Step 9: Make the node point to the next address in the tree
Step 10: Increment the size of the node by 1.
Step 11: If i equals n.size – 1, then update the key of the parent node by using the
         update function.
Step 12: End
```

D) Update :- This function is used to update the value of a node. The algorithm for the function is shown below:

```
Step 1: Start
Step 2: Check if the parent exist or not.
Step 3: if parent.size equals zero, then return the node.
Step 4: Declare an integer variable oldkey that stores the value of old key present in
        the parent node currently.
Step 5: Find the node that contains the old key and replace the old key with the new
        key.
Step 6: End
```

E) Split :- This function is used to split a node when the node is full and we try to enter one more value into that node. The algorithm for the function is shown below:

```
Step 1: Start
Step 2: Declare two integer variables midpoint and newsize and initialise it, also
        declare two objects of node class, i.e., newptr and child.
Step 3: Store the values of the current node to the newptr and change the value of the
        current node.
Step 4: Initialise the node size to midpoint
Step 5: Initialise the new pointer size to newsize.
Step 6: For all the elements in the tree, assign the pointer to next node to child node.
        If child not equals NULL, then child pointer parent node points to the newptr
        node.
Step 7: Return newptr
Step 8: End
```

1) Insert Module :- The key value determines a record's placement in a B+ tree. The leaf pages are maintained in sequential order and a doubly linked list connects each leaf page with its sibling page(s). This doubly linked list speeds data movement as the pages grow and contract.

```
Step 1: Start
Step 2: Input Pharmacy details
Step 3: Check if the tree is empty or not, if it do not exist, then create a root node,
        assign key to the node and increase the node size.
Step 4: If the tree is not empty, then find the leaf node with the help of key and level,
        level is initialised to zero, and using the find leaf function.
Step 5: If node value equals key, then display a message saying that the record already
        exist.
Step 6: Promote function is called to search the correct position where the key can be
        inserted. It has three arguments which is initialised as, initialise n to leaf node,
        key to key and address to NULL.
Step 7: End
```

2) Search Module :- The key to be searched is entered and the for the corresponding the program compares all the index in the file. If there is match in the index then the details are displayed or else an error message is displayed.

```
Step 1: Start
Step 2: Input Pharmacy Id that the user wants to search
Step 3: Check if the tree exist or not before searching the id. If the tree doesn't exist,
        then display that the tree doesn't exist.
Step 4: If the tree is not empty, then find the leaf node with the help of key and level,
        level is initialised to zero and using the find leaf function.
Step 5: Declare a flag variable to keep a track if the node exist and initialise it to zero.
Step 6: If leaf node equals key, then display the Id and increment the flag by 1.
Step 7: If flag equals zero, then display that the tree doesn't exist.
Step 8: End
```

3) Display Module :- All the details stored in a file are displayed in table form.

```
Step 1: Start
Step 2: Create an array of objects of class records
Step 3: Open the text file in input mode and store all the data in records.
Step 4: Create a JTable
Step 5: Insert all the records in a rows of the table.
Step 6: End
```

4) Traversal Module :- The index of all the details is displayed in a tree format, i.e., how the details are stored in file.

```
Step 1: Start
Step 2: Input the pointer to the root node
```

Step 3: If pointer doesn't exist, then return to previous node.

Step 4: Open the text file in output mode and store the value of all the node in that text file.

Step 5: Display the values stored in a file in a tree format using the buffered writer function of java.

Step 6: Repeat from Step 2 with ptr.next[i] as the argument of the function.

4.3 Front End

JFrame form is used to create the front end of the mini project. A frame, implemented as an instance of the JFrame class, is a window that has decorations such as a border, a title, and supports button components that close or iconify the window. Applications with a GUI usually include at least one frame. Applets sometimes use frames, as well.

In the mini project, first there is a login form which allows the login into the system. After login in, the user is transferred to the menu form where he choose the operation that the user wants to perform, corresponding to the option chosen the user has to enter the details that is asked in that form to complete the operation. The user also has the option to go back or logout of the system whenever the user wishes to do so.

If the user chooses the insert option, then the user will be transferred to a page where the user has to enter all the details of the pharmacy and then press save to save the details in the file. If he chooses search option, then he can search for a particular pharmacy detail by entering the pharmacy id. If he chooses the display option, then all the details stored in a file is displayed in table format and lastly, if he chooses the traversal option then he will get the output in a tree format.

4.4 Back End

Java class is used to write and execute the program in the back end. All the functions defined in a class and also called while execution of the form, executes in the back end of the mini project. It is pretty easy to execute and call the program in java, then in any other language.

The different operations used in the program are also written in java class which helps the program for internal operation of the function. Every function has a call to other function internally and java class helps in the proper execution of this function.

CHAPTER 5

RESULTS

In computer systems, a snapshot is the state of a system at a particular point in time. The term was coined as an analogy to that in photography. It can refer to an actual copy of the state of a system or to a capability provided by certain systems.

The snapshots of the form in proper flow is shown below :-

1) Login Form :- In this form, when we enter the username and password the program compares the entered details to the pre-defined value and has two different types of outputs, i.e., if the username and password are correct then the user is allowed to login as shown in Figure 5.1(a) and if the username and password is wrong, then a message is displayed as shown in Figure 5.1(b).

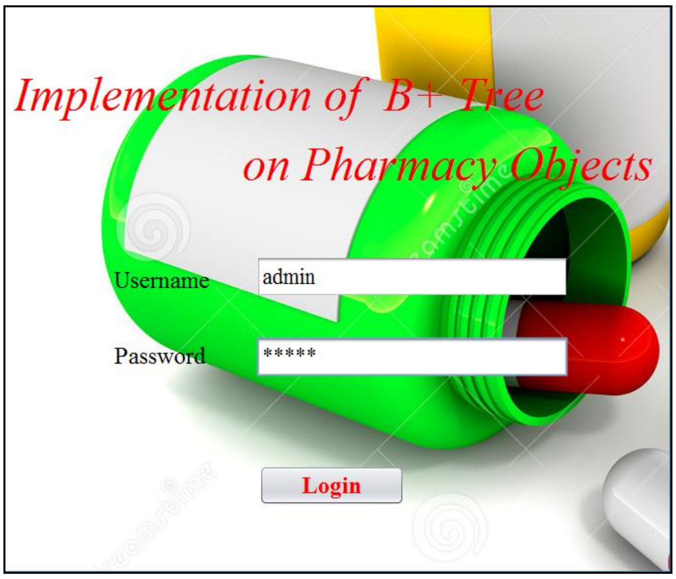
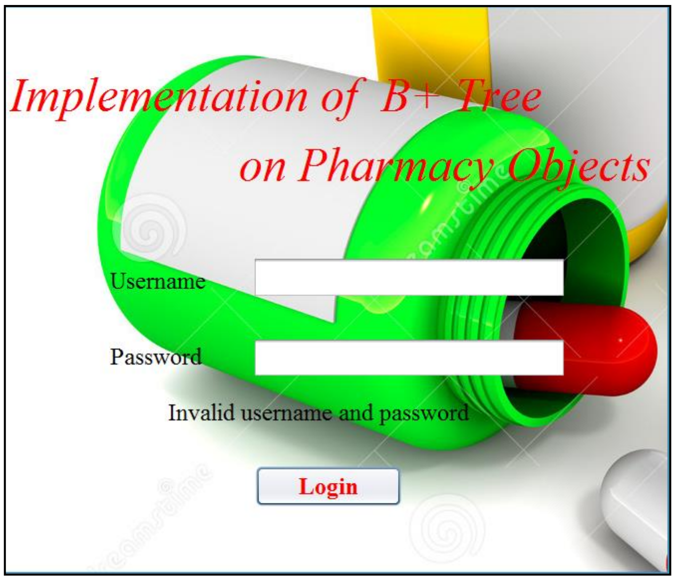


Figure 5.1(a) – Successful Login



Invalid username and password

Figure 5.1(b) – Unsuccessful Login

2) Menu Form :- This forms asks the user to select from any one of the operation listed that the user wants to perform as shown in Figure 5.2 and corresponding to the operation selected the user is transferred to the respective forms.

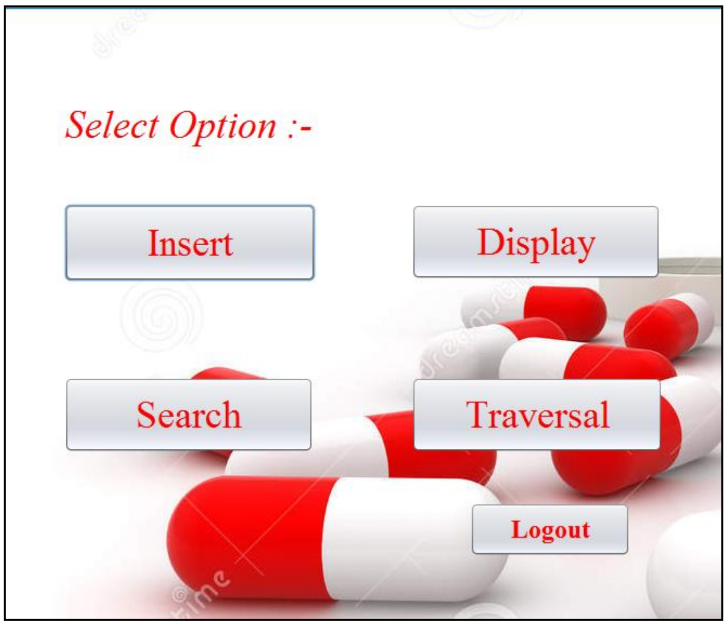
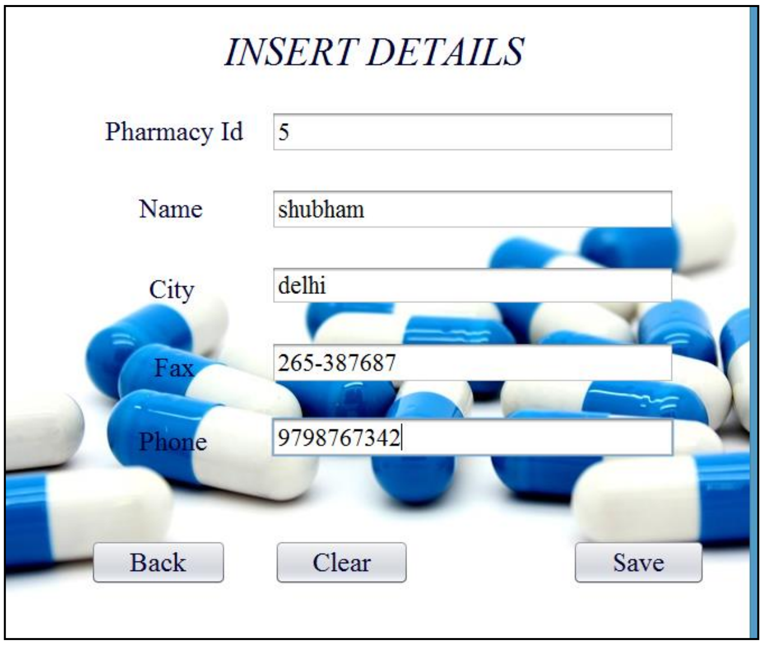


Figure 5.2 – Menu Form

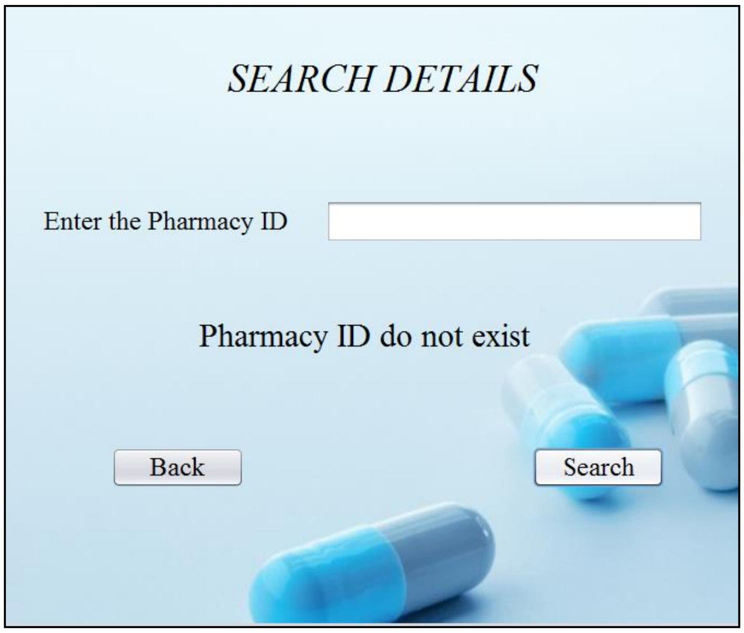
3) Insert Form :- In this form, the user needs to enter all the details that is asked in the form and then click save button to save the details in the file. Figure 5.3 displays successful insertion of pharmacy details into the file. This form calls the insert function in the back end, which reads the input from the fields in the form and then saves it into the file.


INSERT DETAILS

Pharmacy Id	5
Name	shubham
City	delhi
Fax	265-387687
Phone	9798767342

Figure 5.3 – Insert Form

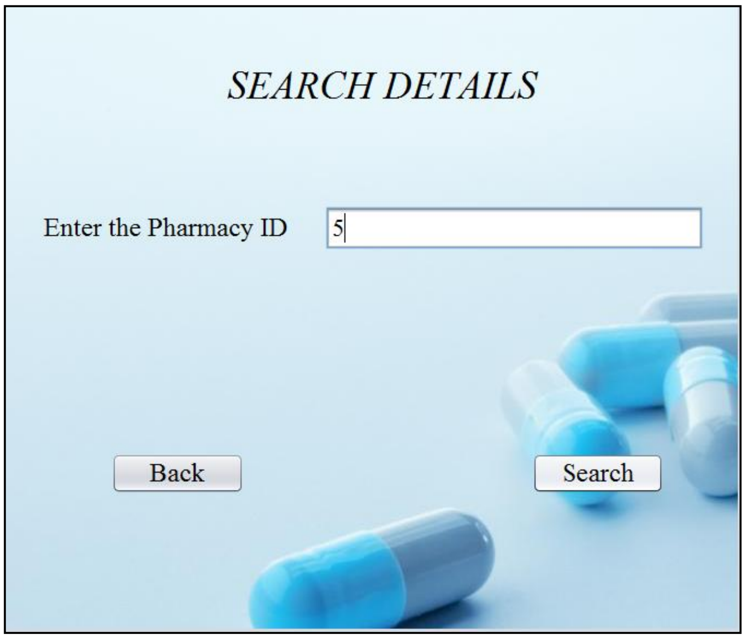
4) Search Form :- This form is used to search for a particular pharmacy information with the help of the unique pharmacy id assigned to it and display the search result as shown in Figure 5.4(a). If the information is not available then it will show an error message as shown in Figure 5.4(b). For the id entered the program calls the search function in the back end, which traverses the file in a tree manner to search for the id and if it finds the id entered, it fetches the details from the files to the front end of the program.


SEARCH DETAILS

Enter the Pharmacy ID

Pharmacy ID do not exist

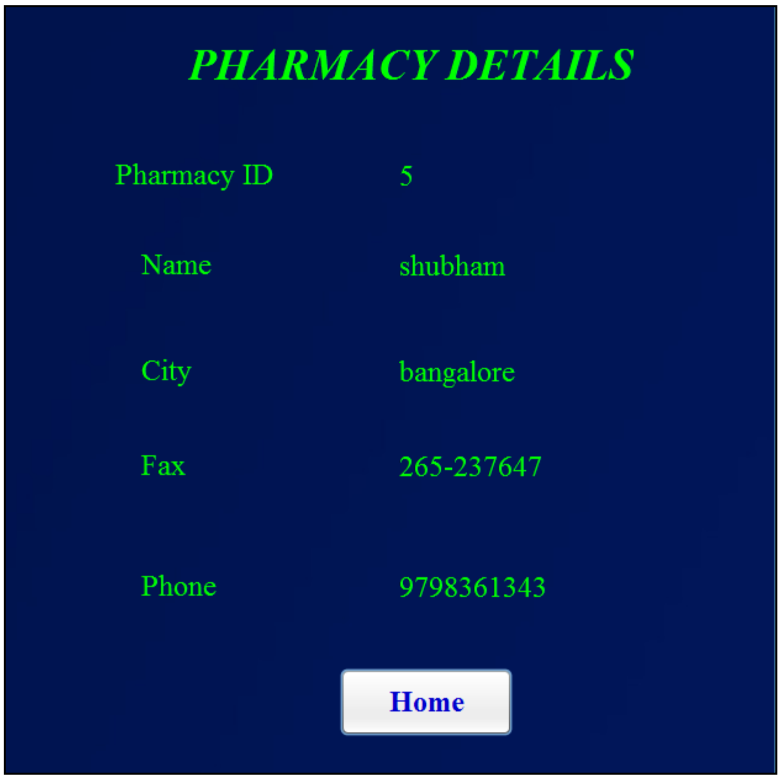
Figure 5.4(a) – Unsuccessful Search


SEARCH DETAILS

Enter the Pharmacy ID 5

Figure 5.4(b) – Successful Search

5) Search Result Form :- This form display's the result of the searched information that the user wants to view as shown in Figure 5.5, by fetching the details from the file.

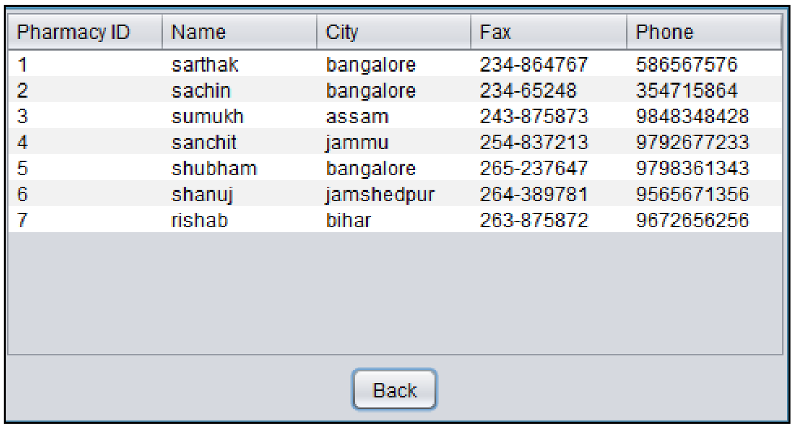


PHARMACY DETAILS

Pharmacy ID	5
Name	shubham
City	bangalore
Fax	265-237647
Phone	9798361343

Figure 5.5 – Search Result

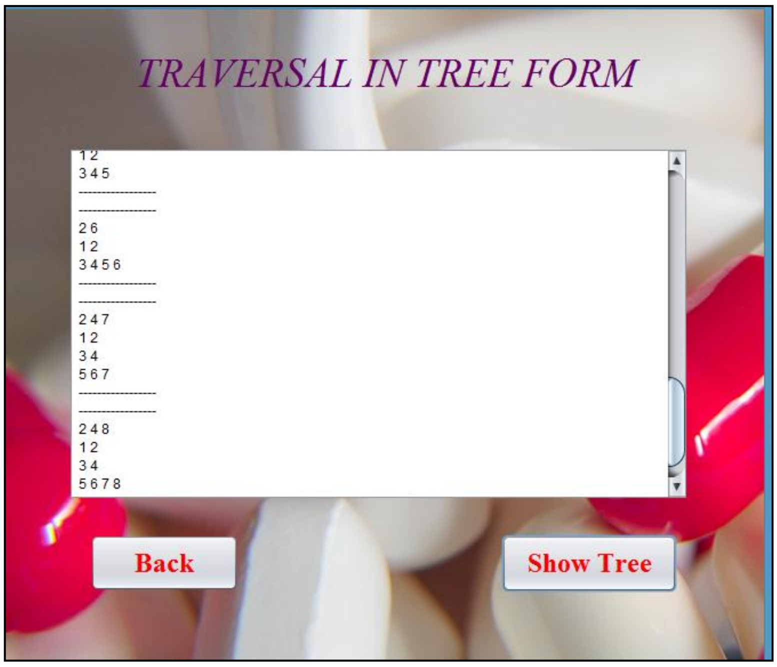
6) Display Form :- This form opens the file and then reads all the information in the file and then displays the list of all the information that is available in the file in the form of a table in the front end as shown in Figure 5.6.



Pharmacy ID	Name	City	Fax	Phone
1	sarthak	bangalore	234-864767	586567576
2	sachin	bangalore	234-65248	354715864
3	sumukh	assam	243-875873	9848348428
4	sanchit	jammu	254-837213	9792677233
5	shubham	bangalore	265-237647	9798361343
6	shanuj	jamshedpur	264-389781	9565671356
7	rishab	bihar	263-875872	9672656256

Figure 5.6 – Display Content

7) Traversal Form :- This form displays the pharmacy id stored in a file in a tree format as shown in Figure 5.7. The tree format used in this program is B+ tree. When the show tree button is clicked, it calls the traverse function in the back end, which arranges all the id in a tree format and displays it in the front end.



```
1 2
3 4 5
---------------
---------------
2 6
1 2
3 4 5 6
---------------
---------------
2 4 7
1 2
3 4
5 6 7
---------------
---------------
2 4 8
1 2
3 4
5 6 7 8
```

Figure 5.7 – Traversal in tree form

CHAPTER 6

CONCLUSION

This mini project provides an interface to the user to store information in a file using B+ tree. This mini project let’s us insert the details of different types of pharmacy that a particular pharmacy is dealing with, it allows the user to search the details of a particular pharmacy that the user wants to know about and it can also be used to review the details of all the pharmacy that a particular is linked up with. It uses a technique know as B+ tree to insert and traverse through the file, as it more efficient and reliable.

The various advantages of B+ tree are as follows :-

1) Short Web Descriptions

B+ tree is an n-array tree with a variable but often large number of children per node. A B+ tree consists of a root, internal nodes and leaves. The root may be either a leaf or a node with two or more children.

2) Storage

In a B+ tree, data is stored in leaf nodes only. The leaf nodes of the tree stores the actual record rather than pointers to records. In leaf nodes data are stored in sequential linked list.

3) Space

B+ Tree do not waste space.

4) Searching

Searching of any data in a B+ tree is very easy because all data is found in leaf nodes.

5) Redundant Key

They store redundant search key.

6) Applications

Many database system implementers prefer the structural simplicity of a B+ tree. There are different applications of B+ tree such as, the ReiserFS, NSS, XFS, JFS, ReFS, and BFS file systems all use this type of tree for metadata indexing; BFS also uses B+ trees for storing directories. NTFS uses B+ trees for directory and security-related metadata indexing. EXT4 uses extent trees (a modified B+ tree data structure) for file extent indexing. Relational database management systems such as IBMDB2, Informix, Microsoft SQL Server, Oracle 8, Sybase ASE, and SQLite support this type of tree for table indices. Key–value database management systems such as CouchDB and Tokyo Cabinet support this type of tree for data access.

One of the limitations of this mini project is that B+ tree is less efficient for static tables. Other limitations is that it does not restricts extra insertion and it supports deletion overhead and space overhead which is also a limitation of B+ tree.

REFERENCES

Textbooks :-

1. Michael J. Folk, Bill Zoellick, Greg Riccardi: File Structures-An Object Oriented Approach with C++, 3rd Edition, Pearson Education, 1998.
2. Jim Keogh: J2EE-TheCompleteReference, McGraw Hill, 2007.

Websites :-

1. https://www.tutorialcup.com/dbms/b-plus-tree.htm
2. https://netbeans.org/kb/trails/matisse.html
3. https://netbeans.org/kb/docs/java/gui-functionality.html
4. www.cburch.com/cs/340/reading/btree/index.html