VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELAGAVI – 590 018

A Mini Project Report on

Covid Patient Record System Using B Tree Indexing Technique

Submitted in partial fulfillment of the requirements as a part of the File Structures Lab of VI semester for the award of degree of ***Bachelor of Engineering*** *in* ***Information Science and Engineering****, Visvesvaraya Technological University, Belagavi*

Submitted by

KAYAK V GORNALE
1RN18IS060

MEHUL GUPTA
1RN18IS066

Faculty Incharge
Mrs. Kusuma S
Assistant Professor
Dept. of ISE, RNSIT

Coordinator
Mr. Santhosh Kumar
Assistant Professor
Dept. of ISE, RNSIT

ESTD : 2001

An Institute with a Difference

Department of Information Science and Engineering

RNS Institute of Technology

Channasandra, Dr. Vishnuvardhan Road, R R Nagar Post
Bengaluru – 560 098

2020 – 2021

RNS Institute of Technology

Channasandra, Dr.Vishnuvardhan Road, RR Nagar Post,
Bengaluru – 560 098

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

ESTD : 2001
An Institute with a Difference

CERTIFICATE

This is to certify that the Mini project report entitled ***COVID PATIENT RECORD SYSTEM USING B TREE INDEXING TECHNIQUE*** has been successfully completed by **KAYAK V GORNALE** bearing USN **1RN18IS060** and **MEHUL GUPTA** bearing USN **1RN18IS066**, presently VI semester students of **RNS Institute of Technology** in partial fulfillment of the requirements as a part of the **File Structure Laboratory** for the award of the degree of ***Bachelor of Engineering in Information Science and Engineering*** under **Visvesvaraya Technological University, Belagavi** during academic year 2020 – 2021. 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 as a part of File structure Laboratory for the said degree.

Mrs. Kusuma S	**Mr. Santhosh Kumar**	**Dr. Suresh L**
Faculty Incharge	Coordinator	Professor and
Assistant Professor	Assistant Professor	Head of Department
Dept. of ISE, RNSIT	Dept. of ISE, RNSIT	Dept. of ISE, RNSIT

External Viva

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

ABSTRACT

File Structures is the Organization of Data in Secondary Storage Device in such a way that it minimizes the access time of the information and the storage space. A File Structure is a combination of representations for data in files and of operations for accessing the data. A File Structure allows applications to read, write and modify data. It might also support finding the data that matches some search criteria or reading through the data in some particular order. The type of data we can store within the file system is Character, Integer, Double, Float, and Boolean.

The Covid Patient Record System is a File structure management system. It provides a user-friendly, interactive Menu Driven Interface (MDI) based on local file systems. All data is stored in files on disk. The application uses file handles to access the files. This project signifies the need for efficient management of patient, their details and test results records in a file. The proposed system enables an administrator to keep track of all the patient records in the File.

In this system, we have a fixed number of fields, each with a maximum length, that combine to make a data record and variable length record. Fixing the number of fields in a record does not imply that the size of fields in the record is fixed. The records are used as containers to hold a mix of fixed and variable-length fields within a record.

Indexing used in this project is B tree. A B-tree is a specialized M-way tree that can be widely used for disk access. A B-Tree of order of m can have at most m-1 keys and m children. One of the main reasons of using B Tree is its capability to store large number of keys in a single node and large number of keys by keeping the height of the tree relatively small.

The primary value of a B tree is in storing data for efficient retrieval in a block- oriented storage context — in particular, filesystems. This is primarily because unlike binary search trees, B trees have very high fanout which reduces the number of I/O operations required to find an element in the tree.

ACKNOWLEDGMENT

The fulfillment and rapture that go with the fruitful finishing of any assignment would be inadequate without the specifying the people who made it conceivable, whose steady direction and support delegated the endeavors with success.

We would like to profoundly thank **Management** of **RNS Institute of Technology** for providing such a healthy environment to carry out this Project work.

We would like to thank our beloved Director **Dr. H N Shivashankar** for his confidence feeling words and support for providing facilities throughout the course.

We would like to express our thanks to our Principal **Dr. M K Venkatesha** for his support and inspired me towards the attainment of knowledge.

We wish to place on record our words of gratitude to **Dr. Suresh L,** Professor and Head of the Department, Information Science and Engineering, for being the enzyme and master mind behind our Project work.

We would like to express our profound and cordial gratitude to our Lab In charge **Mr. Santhosh Kumar**, Assistant Professor, Department of Information Science and Engineering for his valuable guidance, constructive comments and continuous encouragement throughout the Project work.

We would like to express our profound and cordial gratitude to our Faculty In charge Mrs. Kusuma S, Assistant Professor, Department of Information Science and Engineering for his valuable guidance in preparing Project report.

We would like to thank all other teaching and non-teaching staff of Information Science & Engineering who have directly or indirectly helped me to carry out the project work.

And lastly, we would hereby acknowledge and thank our parents who have been a source of inspiration and also instrumental in carrying out this Project work.

KAYAK V GORNALE
USN: 1RN18IS060

MEHUL GUPTA
USN: 1RN18IS066

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

GUI - Graphic User Interface

AVL - Adelson-Velskii and Landis

RAM – Random Access Memory

API – Application Programming Interface

Chapter 1

INTRODUCTION

1.1 Introduction to File Structures

File Structure is a combination of representations of data in files and operations for accessing the data. It allows applications to read, write and modify data. It supports finding the data that matches some search criteria or reading through the data in some particular order.

1.1.1 History

Early work with files presumed that files were on tape, since most files were. Access was sequential and the cost of access grew in direct proportion to the size of the file. As files grew intolerably large for unaided sequential access and as storage devices such as hard disks became available, indexes were added to files.

The indexes made it possible to keep a list of keys and pointers in a smaller file that could be searched more quickly. With key and pointer, the user had direct access to the large, primary file. But simple indexes had some of the same sequential flaws as the data file, and as the indexes grew, they too became difficult to manage, especially for dynamic files in which the set of keys changes.

In the early 1960's, the idea of applying tree structures emerged. But trees can grow very unevenly as records are added and deleted, resulting in long searches requiring many disk accesses to find a record. In 1963, researchers developed an elegant, self-adjusting binary tree structure, called AVL tree, for data in memory.

The problem was that, even with a balanced binary tree, dozens of accesses were required to find a record in even moderate sized files. A method was needed to keep a tree balanced when each node of the tree was not a single record, as in a binary tree, but a file block containing dozens, perhaps even hundreds, of records. It took 10 years until a solution emerged in the form of a B-tree.

Whereas AVL trees grow from the top down as records were added, B-trees grew from the bottom up. B-trees provided excellent access performance, but there was a cost: no longer could a file be accessed sequentially with efficiency. The problem was solved by adding a linked list structure at the bottom level of the B-tree.

The combination of a B-tree and a sequential linked list is called a B+ tree. B-trees and B+ trees provide access times that grow in proportion to log k N, where N is the number of entries in the file and k is the number of entries indexed in a single block of the B-tree structure. This means that B-trees can guarantee that you can find 1 file entry among millions with only 3 or 4 trips to the disk. Further, B-trees guarantee that as you add and delete entries, performance stays about the same.

Hashing is a good way to get what we want with a single request, with files that do not change size greatly over time. Hashed indexes were used to provide fast access to files. But until recently, hashing did not work well with volatile, dynamic files. Extendible dynamic hashing can retrieve information with 1 or at most 2 disk accesses, no matter how big the file became.

1.1.2 About the File

There is one important distinction that in 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.

A file system consists of two or three layers. Sometimes the layers are explicitly separated, and sometimes the functions are combined. The logical file system is responsible for interaction with the user application.

It provides the application program interface (API) for file operations- open, close, read etc., and passes the requested operation to the layer below it for processing. The logical file system "manages open file table entries and per-process file descriptors. This layer provides file access, directory operations, and security and protection.

The second optional layer is the virtual file system. "This interface allows support for multiple concurrent instances of physical file systems, each of which is called a file system implementation.

The third layer is the physical file system. This layer is concerned with the physical operation of the storage device (e.g. disk). It processes physical blocks being read or written. It handles buffering and memory management and is responsible for the physical placement of blocks in specific locations on the storage medium. The physical file system interacts with the device drivers or with the channel to drive the storage device.

1.1.3 Applications of File Structure

Relative to other parts of a computer, disks are slow. One can pack thousands of megabytes on a disk that fits into a notebook computer. The time it takes to get information from even relatively slow electronic RAM is about 120 ns. Getting the same information from a typical disk takes 30ms. So, the disk access is quarter of a million times longer than a memory access. Hence, disks are very slow compared to memory.

On the other hand, disks provide enormous capacity at much less cost than memory. They also keep the information stored on them when they are turned off. Tension between a disk's relatively slow access time and its enormous, non-volatile capacity is the driving force behind file structure design. Good file structure design will give us access to all the capacity without making our applications spend a lot of time waiting for the disk.

Chapter 2

SYSTEM ANALYSIS

Systems analysis is the process of observing systems for troubleshooting or development purposes. It is applied to information technology, where computer-based systems require defined analysis according to their makeup and design.

2.1 Analysis of the project

The application deals with the covid patients record, which keeps the records of all the tested patients. Using this application, we can add new patient details or modify information of existing patient details. The applications give us the options to display all the positive or all the patient records along with the dates when they were tested positive or negative. This application implements B-tree indexing technique. The application even provides us the option to show the B-Tree structure.

2.2 Structure used to store the fields and records

2.2.1 Field Structure

A field is the smallest, logically meaningful, unit of information in a file. Field Structures: Four of the most common methods of adding structure to files to maintain the identity of fields are:

- Force the fields into a predictable length.
- Begin each field with a length indicator.
- Place a delimiter at the end of each field to separate it from the next field.
- Use a “keyword=value” expression to identify each field and its contents.

Method 1: Fix the Length of Fields

Force the fields into a predictable length. This method relies on creating fields of predictable fixed size.

Disadvantages:

a. A lot of wasted space due to padding of fields with whitespace or empty space
b. Data values may not fit in the pre-determined length if the fields.

The fixed-length field approach is inappropriate for data that inherently contains a large amount of variability in the lengths of fields such as names or addresses.

Method 2: Begin Each Field with a Length Indicator

We can count to the end of a field by storing the field length just ahead of the field. If the fields are not too long (less than 256 bytes), it is possible to store the length in a single byte at the start of each field. We refer to these fields as length-based.

Method 3: Separate the Fields with Delimiters

This method requires that the fields be separated by a selected special character or a sequence of characters called a delimiter. Ex: If "|" is used as delimiter then a sample record would look like: Ames|Mary|123Maple|StillWater|OK|574075

This method of separating fields with a delimiter is often used. However, choosing a right delimiter is very important. In many cases white space characters are excellent delimiters because they provide a clean separation between the fields when they are listed on the console.

Method 4: Use a "Keyword=Value" Expression to Identify Fields

This method requires that each field data be preceded with the field identifier/ keyword. It can also be used with the combination of delimiter to mark the field ends. Ex: last=Ames|first=Mary|address=123Maple|city=StillWater|state=ok|zip=570405

Some of the advantages are:

1. Each field provides information about itself.
2. Good format for dealing with missing fields.

The disadvantage here is that in some applications, a lot of space may be wasted on field keywords.

2.2.2 Record Structures:

The five most often used methods for organizing records of a file are:

- Require the records to be predictable number of bytes in length.
- Require the records to be predictable number of fields in length.
- Begin each record with a length indicator consisting of a count of the number of bytes that the record contains.
- Use a second file to keep track of the beginning byte address for each record.
- Place a delimiter at the end of each record to separate it from the next record

Method 1: Make the Records a Predictable Number of Bytes (Fixed-Length-Record)

A fixed-length record file is one in which each record contains the same number of bytes. In the field and record structure shown, we have a fixed number of fields, each with a predetermined length, that combine to make a fixed-length record. Fixing the number of bytes in a record does not imply that the size or number of fields in the record must be fixed.

Fixed-length records are often used as containers to hold variable numbers of variable-length fields. It is also possible to mix fixed and variable length fields within a record.

Method 2: Make Records a Predictable Number of Fields

This method specifies the number of fields in each record. Regardless of the method for storing fields, this approach allows for relatively easy means of calculating record boundaries.

Method 3: Begin Each Record with a Length Indicator

We can communicate the length of records by beginning each record with a filed containing an integer that indicates how many bytes there are in the rest of the record containing an integer that indicates how many bytes there are in the rest of the record. This is commonly used to handle variable-length records.

Method 4: Use an Index to Keep Track of Addresses

We can use an index to keep a byte offset for each record in the original file. The byte offset allows us to find the beginning of each successive record and compute the length of each record. We look up the position of a record in the index, then seek to the record in the data file.

Method 5: Place a Delimiter at the End of Each Record

It is analogous to keeping the fields distinct. As with fields, the delimiter character must not get in the way of processing. A common choice of a record delimiter for files that contain readable text is the end-of- line character (carriage return/ new-line pair or, on Unix systems, just a new- line character: \n). Here, we use a # character as the record delimiter.

2.3 Operation Performed on a File

- **Insertion**

  Insertion function adds new records into the file. In this process the inserted node descends to the leaf where the key fits. If the node has an empty space, insert the key/reference pair into the node. If the node is already full, split it into two nodes, distributing the keys evenly between the two nodes.

- **Deletion**

  Deletion is the process in which on the parent node to remove the split key that previously separated these merged nodes unless the parent is the root and we are removing the final key from the root, in which case the merged node becomes the new root.

- **Update**

  The reference to a deleted record is removed from index while the deleted record persists in the data file. A '$' is placed in the first byte of the first field (**ID**) of the record, to help distinguish it from records that should be displayed. The requested record, if found, is marked for deletion, a "record deleted" message is displayed, and the reference to it is removed from the B+ Tree. If absent, a "record not found" message is displayed to the user. If present, after deletion, the system is used to insert the modified record containing, possibly, a new set of data like name, address, occupation it can be updated.

- **Search**

  Searching starts 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 whose range includes the search value. A subtree'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 used within nodes to find the separation values and child tree of interest. If the key matches, then the details of the record are displayed on the console else we continue to search. If the record is not found then we return -1 value and display 'RECORD DOES NOT EXISTS' message on the console.

- **Modify**

  The system can also be used to modify existing records from all production inventory details. The user is prompted for a key, which is used as to search the records. Once the record is found we rewrite details of the corresponding product Hence modify () is delete () followed by the read (). If the record is not found then it returns -1 value and display 'RECORD DOES NOT EXISTS' message on the console.

2.4 Indexing Used

A B-Tree indexes uses a primary key and a pointer pointing to the data file. All leaves are at the same level. A B-Tree is defined by the term minimum degree 't'. The value of t depends upon disk block size. Every node except root must contain at least (ceiling)([t-1]/2) keys. The root may contain minimum 1 key. All nodes (including root) may contain at most t – 1 key. Number of children of a node is equal to the number of keys in it plus 1. All keys of a node are sorted in increasing order. The child between two keys k1 and k2 contains all keys in the range from k1 and k2. B-Tree grows and shrinks from the root which is unlike Binary Search Tree. Binary Search Trees grow downward and also shrink from downward. Like other balanced Binary Search Trees, time complexity to search, insert and delete is O(log n). Insertion of a Node in B-Tree happens only at Leaf Node

Chapter 3

SYSTEM DESIGN

The System Design Document describes the system requirements, operating environment, system and subsystem architecture, files and database design, input formats, output layouts, human-machine interfaces, detailed design, processing logic, and external interfaces. Systems design is the process of defining the architecture, modules, interfaces, and data for a system to satisfy specified requirements. Systems design could be seen as the application of systems theory to product development.

3.1 Design of the Fields and records

We use variable length record and fixed size fields. In variable length record, it will take the minimum space need for that field and at the end of the record there will be a delimiter placed indicating that it is the end of the record. The fields are separated using a delimiter (‘|’) whereas hash (‘#’) denotes end of a record.

A delimiter is a sequence of one or more characters used to specify the boundary between separate, independent regions in plain text or other data streams. In this application, there are 3 files which stores the data they are:

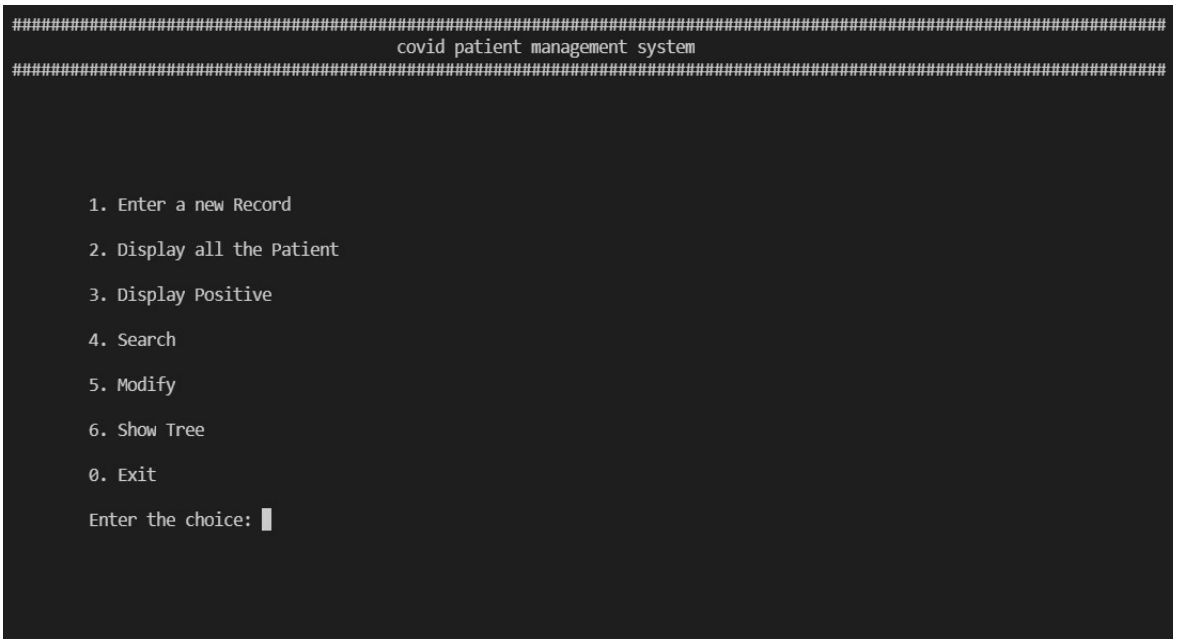
1. Data File
2. Index File

These files have respective attributes which identifies the uniqueness of each record.

1. Data File: It consists of the all the patients details which contains Aadhar Number, Name, Age, Gender, Phone Number, City, Date of positive and negative, Result.
2. Index File: It consist two fields : primary key which is Aadhar Number in the Data file and the address to the corresponding record in the Data file.

3.2 User Interface

The user interface (UI), in the industrial design field of human–computer interaction, is the space where interactions between humans and machines occur. The goal of this interaction is to allow effective operation and control of the machine from the human end, whilst the machine simultaneously feeds back information that aids the operators' decision-making process.



```
########################################################################################################################
                                              covid patient management system
########################################################################################################################



        1. Enter a new Record

        2. Display all the Patient

        3. Display Positive

        4. Search

        5. Modify

        6. Show Tree

        0. Exit

        Enter the choice: █
```

Figure 3.1 User Interface

3.2.1 Insertion of a record

If the user wants to enter a new record he/she should choose option 1 from the menu, then he/she is prompted to a new screen where he/she is required to enter the following data:

- Aadhar Number: Here the Aadhar Number of new Patient is entered and it should be of 12 digits and no character and it should not be already present in the Data file.
- Name: Enter the patient’s name.
- Age: Enter the patient’s age.
- Gender: Enter the patient’s gender (M/F/O)
- Mobile Number: Enter the patient’s mobile number.
- City: Enter the city where the patient got tested.
- Date of positive: Enter the date when the patient was tested covid-19 positive.
- Result: Enter the patient’s covid-19 test result (P/N)
- Date of Negative: Enter the date when the patient was tested covid-19 negative.

And this data is stored in the data file and the Aadhar number and it's corresponding address in the data file is stored in the B-Tree and Index File.

3.2.2 Display of all patients record

Here the records of all the patients are displayed from the Data file. Here the records are displayed in the ascending order with respect to the Aadhar number. For displaying we traverse the B-Tree which consists of the Aadhar number and a pointer which points to corresponding record in the Data file. In the beginning we construct the B-Tree using existing values in the Index file and after each operation we update the Index file with the help of B-Tree values.

3.2.3 Display of only positive patient's record

Here the records of only positive patients are displayed from the Data file. Here the records are displayed in the ascending order with respect to the Aadhar number. For displaying we traverse the B-Tree which consists of the Aadhar number and a pointer which points to corresponding record in the Data file and whose Result field is 'P'.

3.2.4 Searching of a record

If the operation desired is Search, the user is required to enter option 4 as his/her choice, from the menu displayed, after which a new screen is displayed. Next, the user is prompted to enter the Aadhar Number (key) that has to be searched. If the record is not found in the data file, a "RECORD NOT FOUND" message is displayed, and the user is prompted to press 0 to return back to themenu screen. If the matching key is found, the corresponding record is to be retrieved, with details of the record that was to be searched. In each case, the user is then prompted to press 0 to return back to the menu screen.

3.2.5 Modification of a Record

If the user chooses option 5 which is to modify, the user is required to enter the Aadhar Number which is to be modified, If the entered Aadhar number doesn't exist in the Data File, a "Record Not Found" message is displayed and the user needs to press 0 to return back to the main menu. If the record exists then the respective node is deleted from the B-Tree and then user is prompted to a new screen where user needs to enter the complete and updated record and then it is re-inserted to the B-Tree with new address and user is prompted to enter 0 to go back to main menu.

Chapter 4

IMPLEMENTATION

Implementation is the process of defining how the system should be built, ensuring that it is operational and meets quality standards. It is a systematic and structured approach for effectively integrating a software-based service or component into the requirements of end users.

4.1 About C++

C++ is a general-purpose object-oriented programming (OOP) language, developed by Bjarne Stroustrup, and is an extension of the C language. It is therefore possible to code C++ in a "C style" or "object-oriented style." In certain scenarios, it can be coded in either way and is thus an effective example of a hybrid language.

C++ is considered to be an intermediate-level language, as it encapsulates both high- and low-level language features. Initially, the language was called "C with classes" as it had all the properties of the C language with an additional concept of "classes." However, it was renamed C++ in 1983.

The main highlight of C++ is a collection of predefined classes, which are data types that can be instantiated multiple times. The language also facilitates declaration of user-defined classes. Classes can further accommodate member functions to implement specific functionality. Multiple objects of a particular class can be defined to implement the functions within the class. Objects can be defined as instances created at run time. These classes can also be inherited by other new classes which take in the public and protected functionalities by default.

C++ includes several operators such as comparison, arithmetic, bit manipulation and logical operators. One of the most attractive features of C++ is that it enables the overloading of certain operators such as addition. A few of the essential concepts within the C++ programming language include polymorphism, virtual and friend functions, templates, namespaces and pointers.

4.2 Pseudocode

Pseudo code is a detailed yet readable which represents the programming language statements for each module which is easier to understand of our application and its main idea is to give out its functionality.

4.2.1 Insertion Module Pseudocode

The following pseudocode represents the insertion module.

```
1. Initialize x as root.
2. While x is not leaf, do following
      a. Find the child of x that is going to be traversed next. Let the child be y.
      b. If y is not full, change x to point to y.
      c. If y is full, split it and change x to point to one of the two parts of y. If k
   is smaller than mid key in y, then set x as the first part of y. Else second part of
   y. When we split y, we move a key from y to its parent x.
3. The loop in step 2 stops when x is leaf. x must have space for 1 extra key as we
   have been splitting all nodes in advance. So simply insert k to x.
```

4.2.2 Display Module Pseudocode

The following pseudocode represents the display module.

```
1. Open the datafile.
2. Check if datafile is empty; If empty break. Else continue the execution.
3. Set the indentation to left using setiosflags.
4. Retrieve the address of the first field in the first record.
5. While until the condition is true
      a. Unpacks the fields of the first record.
      b. Displays the contents of the record.
6. Increments the pointer to the next record.
7. Exits the loop when the file pointer reaches end of file
```

4.2.3 Delete Module Pseudocode

The following algorithm represents the deletion module pseudocode. We sketch how deletion works with various cases of deleting keys from a B-Tree.

```
1. If the key k is in node x and x is a leaf, delete the key k from x.
2. If the key k is in node x and x is an internal node, do the following.
      a. If the child y that precedes k in node x has at least t keys, then find
         the predecessor k0 of k in the sub-tree rooted at y. Recursively
         delete k0, and replace k by k0 in x. (We can find k0 and delete it in
         a single downward pass.)
      b. If y has fewer than t keys, then, symmetrically, examine the child z
         that follows k in node x. If z has at least t keys, then find the
         successor k0 of k in the subtree rooted at z. Recursively delete k0,
         and replace k by k0 in x. (We can find k0 and delete it in a single
         downward pass.)
      c. Otherwise, if both y and z have only t-1 keys, merge k and all of z
         into y, so that x loses both k and the pointer to z, and y now
         contains 2t-1 keys. Then free z and recursively delete k from y.

3. If the key k is not present in internal node x, determine the root x.c(i) of
   the appropriate subtree that must contain k, if k is in the tree at all. If x.c(i)
   has only t-1 keys, execute step 3a or 3b as necessary to guarantee that we
   descend to a node containing at least t keys. Then finish by recursing on
   the appropriate child of x.
      a. If x.c(i) has only t-1 keys but has an immediate sibling with at least
         t keys, give x.c(i) an extra key by moving a key from x down into
         x.c(i), moving a key from x.c(i) 's immediate left or right sibling up
         into x, and moving the appropriate child pointer from the sibling
         into x.c(i).
      b. If x.c(i) and both of x.c(i)'s immediate siblings have t-1 keys,
         merge x.c(i) with one sibling, which involves moving a key from x
         down into the new merged node to become the median key for that
         node.
```

4.2.4 Search Module Pseudocode

In B-Tree, a search is one of the easiest procedures to execute and get fast and accurate results from it.

```
1. Call the binary search method on the records in the B-Tree.
2. If the search parameters match the exact key
      a. The accurate result is returned and displayed to the user
 Else
   1. if the node being searched is the current and the exact key is not found by
      the algorithm
   2. Display the statement "Record set cannot be found."
```

4.3 Testing

4.3.1 Unit Testing

Unit testing is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output.

Unit testing is commonly automated, but may still be performed manually. The objective in unit testing is to isolate a unit and validate its correctness. A manual approach to unit testing may employ a step-by-step instructional document. The unit testing is the process of testing the part of the program to verify whether the program is working correct or not. In this part the main intention is to check the each and every input which we are inserting to our file. Here the validation concepts are used to check whether the program is taking the inputs in the correct format or not.

Unit testing may reduce uncertainty in the units themselves and can be used in a bottom-up testing style approach. By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier. Unit test cases embody characteristics that are critical to the success of the unit. This testing is demonstrated in the table 4.1

Table 4.1 Unit testing

Case ID	Description	Input Data	Expected Output	Actual Output	Status
1	Opening a file insert the data	Insert option is selected	File should be opened in append mode without any error messages	File opened in a append mode	Pass
2	Validating Aadhar number	123456789123	Accept the Aadhar Number	"Enter the Name"	Pass
3	Validating Aadhar number	123876	Invalid Aadhar Number	"Aadhar Number should have 12 digits and no characters"	Pass
4	Validating Aadhar number	1235435A43	Invalid Aadhar Number	"Aadhar Number should have 12 digits and no characters"	Pass
5	Validating Aadhar number	123456789123	Aadhar Number already exists	"This aadhar already exists. You can choose option 5 modify to edit the info."	Pass

6	Validating the phone number	9758923148	It should accept the phone number	It continues taking information	Pass
7	Validating the phone number	9893542a	Invalid Phone number	"Phone number should be 10 digits only and no characters."	Pass
8	File updating	-	It should pack all the fields and will be appended to the file	Data will be updated both in data and index file	Pass
9	Closing a File	-	File should be closed without any error	File is closed	pass
10	Date Format validation	10-45-2019	Invalid data "Enter the correct date format"	"Enter the correct data format"	Pass

4.3.2 Integration Testing

Integration testing is also taken as integration and testing this is the major testing process where the units are combined and tested. Its main objective is to verify whether the major parts of the program is working fine or not. This testing can be done by choosing the options in the program and by giving suitable inputs it is tested and is shown in table 4.2

Table 4.2 Integration Testing

Case ID	Description	Input Data	Expected Output	Actual Output	Status
1	To display the entered records of the data file	Enter the option 2 in the menu	Display all the records one after the other	Display all the records one after the other	Pass

2	To add the new records into the data file	Enter the option 1. In the menu	Display the record entry form	Display the record entry form	Pass
3	To search for a particular record in the file	Enter the option 4 in the menu and should enter the Aadhar Number: 123456789123	Record not found.	Record not found.	Pass
4	To search for all particular record in the file.	Enter the option 4 in the menu and should enter the Aadhar Number: 123456789123	'Record is found' and it will display the contents of the searched record	Same as expected output.	Pass
5	To display the positive records of the data file	Enter the option 3 in the menu	Display all the Positive records one after the other	Display all the Positive records one after the other	Pass
6	To display the B+ tree of all record in the file	Enter the option 6 in the menu	Displays the B-Tree of the data file record	Displays the B-Tree of the data file record	Pass
7	To quit the program	Enter the option 6 in the menu	Exit the program Pass	Exit the program Pass	Pass

4.3.3 System Testing

System testing is defined as testing of a complete and fully integrated software product. This testing falls in black-box testing wherein knowledge of the inner design of the code is not a pre-requisite and is done by the testing team. System testing is done after integration testing is complete. System testing should test functional and non-functional requirements of the software. And is implemented in below table 4.3.

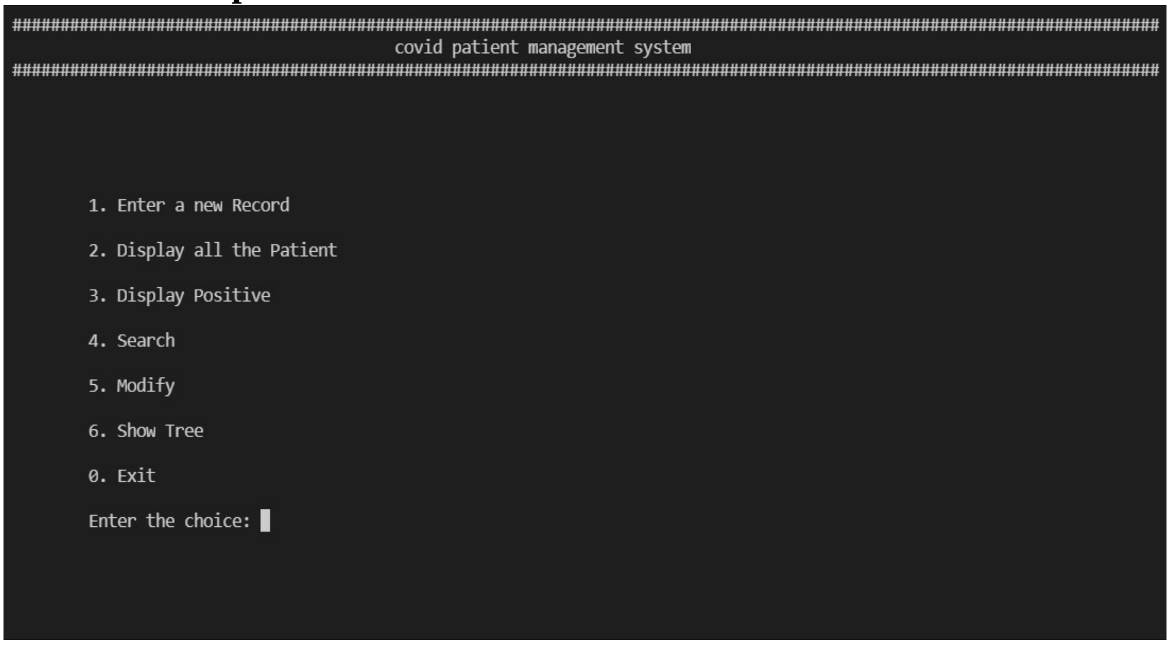
Table 4.3 System Testing

Case ID	**Description**	**Input Data**	**Expected Output**	**Actual Output**	**Status**
1	To display the records	Display records for valid Aadhar Number = 123456789123(present) and invalid Aadhar Number = 1234564M (not present)	Record is displayed for valid and record not found for invalid	Record is displayed for valid and record not found for invalid	Pass
2	To search the record	Search records for valid Aadhar Number = 123456789123(present) and invalid Aadhar Number = 1234564M (not present)	Record is displayed for valid and record not found for invalid	Record is displayed for valid and record not found for invalid	pass
3	To modify the records	Modify records for valid Aadhar Number =123456789123(present) and invalid Aadhar Number = 1234564M (not present)	Record is modified for valid and record not found for invalid	Record is modified for valid and record not found for invalid	Pass

4.4 Discussion of Results

All the menu options provided in the application and its operations have been presented in as snapshots. A detailed view of all the snapshots of the output screen is given here.

4.4.1 Menu Options



```
#####################################################################################################################
                                         covid patient management system
#####################################################################################################################




        1. Enter a new Record

        2. Display all the Patient

        3. Display Positive

        4. Search

        5. Modify

        6. Show Tree

        0. Exit

        Enter the choice:
```

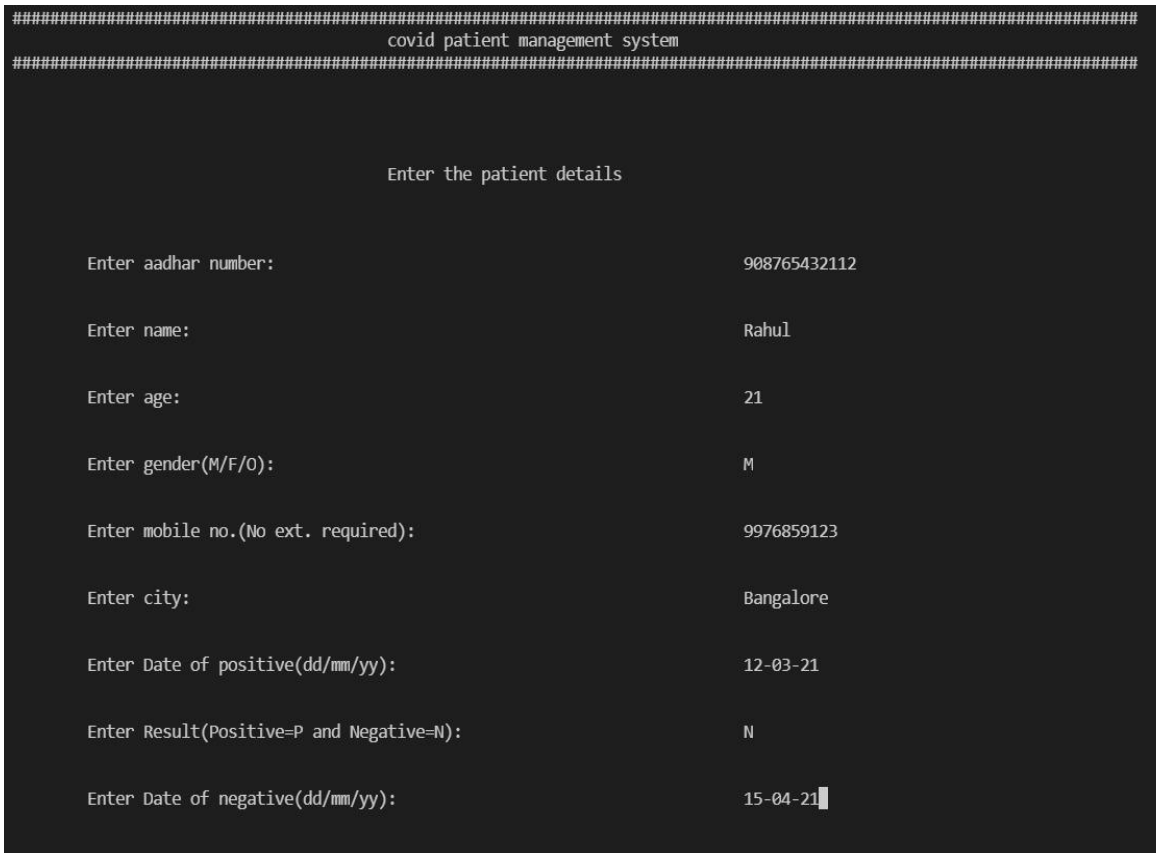
Figure 4.1 Menu Options

The Figure 4.1 snapshot shows the menu which has the number of option and giving suggestion to the user of the system.

The menu of the application in which the user has to give his input depending on the options available, the user can perform any of the operations such as Read operations, Display All Operations, Search Operations and last exit where user can end program process.

4.4.2 Read Operation

Read Operation takes a new patient's information and saves it as a record in the Data file.



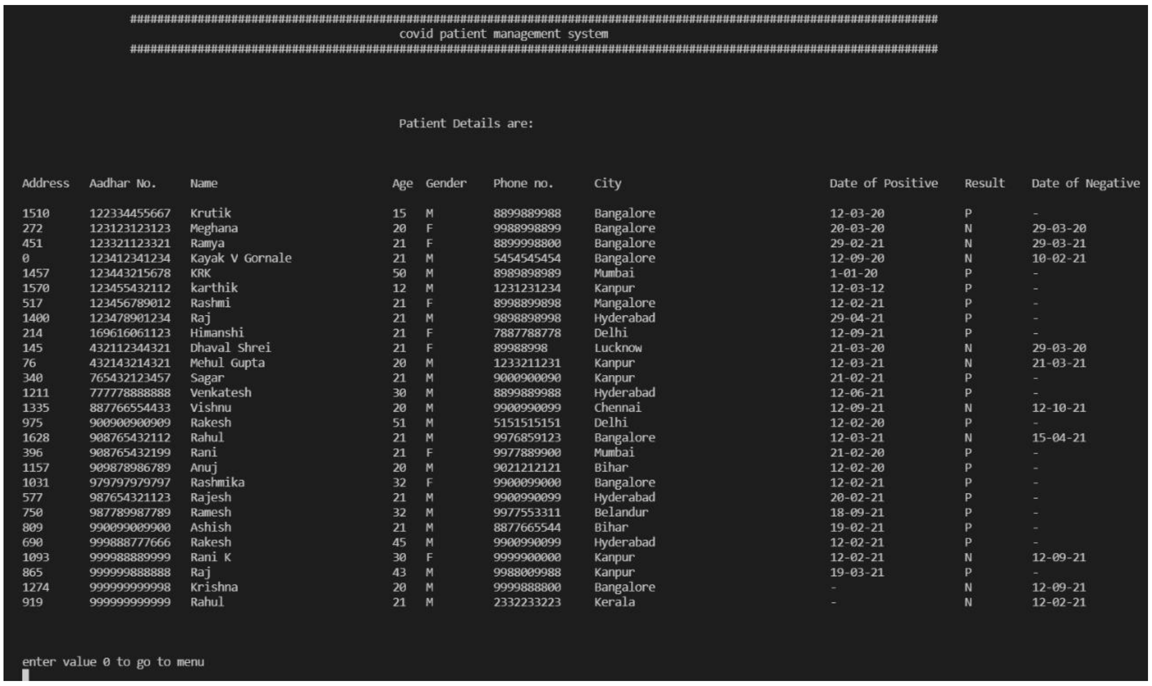
```
########################################################################################################################
                                        covid patient management system
########################################################################################################################



                                        Enter the patient details


        Enter aadhar number:                                                    908765432112

        Enter name:                                                             Rahul

        Enter age:                                                              21

        Enter gender(M/F/O):                                                    M

        Enter mobile no.(No ext. required):                                     9976859123

        Enter city:                                                             Bangalore

        Enter Date of positive(dd/mm/yy):                                       12-03-21

        Enter Result(Positive=P and Negative=N):                                N

        Enter Date of negative(dd/mm/yy):                                       15-04-21
```

Figure 4.2 Read Operation

Insertion of a new record and new corresponding Aadhar Number and its address is stored in Data file and Index file respectively.

4.4.3 Display All Operation

Display all Operations displays every covid-19 tested patient record in sorted order according to their Aadhar number by traversing the B-Tree.

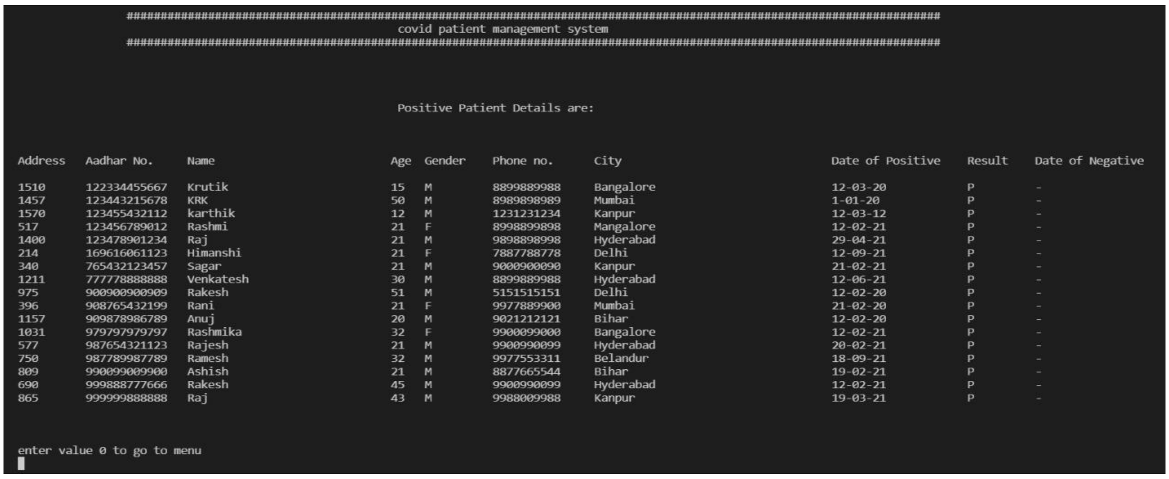


```
          ########################################################################################################
                                             covid patient management system
          ########################################################################################################



                                              Patient Details are:


Address  Aadhar No.    Name                         Age  Gender    Phone no.     City                              Date of Positive    Result    Date of Negative

1510     122334455667  Krutik                       15   M         8899889988    Bangalore                         12-03-20            P         -
272      123123123123  Meghana                      20   F         9988998899    Bangalore                         20-03-20            N         29-03-20
451      123321123321  Ramya                        21   F         8899998800    Bangalore                         29-02-21            N         29-03-21
0        123412341234  Kayak V Gornale              21   M         5454545454    Bangalore                         12-09-20            N         10-02-21
1457     123443215678  KRK                          50   M         8989898989    Mumbai                            1-01-20             P         -
1570     123455432112  karthik                      12   M         1231231234    Kanpur                            12-03-12            P         -
517      123456789012  Rashmi                       21   F         8998899898    Mangalore                         12-02-21            P         -
1400     123478901234  Raj                          21   M         9898898998    Hyderabad                         29-04-21            P         -
214      169616061123  Himanshi                     21   F         7887788778    Delhi                             12-09-21            P         -
145      432112344321  Dhaval Shrei                 21   F         89988998      Lucknow                           21-03-20            N         29-03-20
76       432143214321  Mehul Gupta                  20   M         1233211231    Kanpur                            12-03-21            N         21-03-21
340      765432123457  Sagar                        21   M         9000900090    Kanpur                            21-02-21            P         -
1211     777778888888  Venkatesh                    30   M         8899889988    Hyderabad                         12-06-21            P         -
1335     887766554433  Vishnu                       20   M         9900990099    Chennai                           12-09-21            N         12-10-21
975      900900900909  Rakesh                       51   M         5151515151    Delhi                             12-02-20            P         -
1628     908765432112  Rahul                        21   M         9976859123    Bangalore                         12-03-21            N         15-04-21
396      908765432199  Rani                         21   F         9977889900    Mumbai                            21-02-20            P         -
1157     909878986789  Anuj                         20   M         9021212121    Bihar                             12-02-20            P         -
1031     979797979797  Rashmika                     32   F         9900099000    Bangalore                         12-02-21            P         -
577      987654321123  Rajesh                       21   M         9900990099    Hyderabad                         20-02-21            P         -
750      987789987789  Ramesh                       32   M         9977553311    Belandur                          18-09-21            P         -
809      990099009900  Ashish                       21   M         8877665544    Bihar                             19-02-21            P         -
690      999888777666  Rakesh                       45   M         9900990099    Hyderabad                         12-02-21            P         -
1093     999988889999  Rani K                       30   F         9999900000    Kanpur                            12-02-21            N         12-09-21
865      999999888888  Raj                          43   M         9988009988    Kanpur                            19-03-21            P         -
1274     999999999998  Krishna                      20   M         9999888800    Bangalore                         -                   N         12-09-21
919      999999999999  Rahul                        21   M         2332233223    Kerala                            -                   N         12-02-21


enter value 0 to go to menu
█
```

Figure 4.3 Display All Operation

4.4.4 Display Positive Operation

Display positive Operation displays every covid-19 positive patient record in sorted order according to their Aadhar number by traversing the B-Tree.

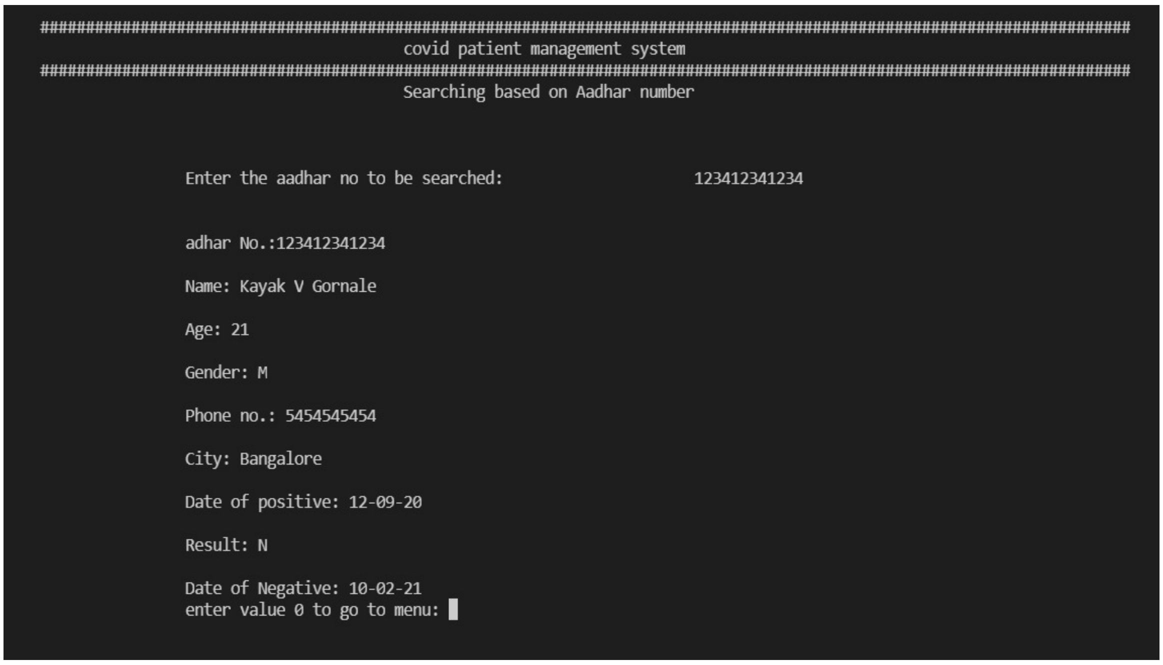


```
          ########################################################################################################
                                             covid patient management system
          ########################################################################################################



                                              Positive Patient Details are:


Address  Aadhar No.    Name                         Age  Gender    Phone no.     City                              Date of Positive    Result    Date of Negative

1510     122334455667  Krutik                       15   M         8899889988    Bangalore                         12-03-20            P         -
1457     123443215678  KRK                          50   M         8989898989    Mumbai                            1-01-20             P         -
1570     123455432112  karthik                      12   M         1231231234    Kanpur                            12-03-12            P         -
517      123456789012  Rashmi                       21   F         8998899898    Mangalore                         12-02-21            P         -
1400     123478901234  Raj                          21   M         9898898998    Hyderabad                         29-04-21            P         -
214      169616061123  Himanshi                     21   F         7887788778    Delhi                             12-09-21            P         -
340      765432123457  Sagar                        21   M         9000900090    Kanpur                            21-02-21            P         -
1211     777778888888  Venkatesh                    30   M         8899889988    Hyderabad                         12-06-21            P         -
975      900900900909  Rakesh                       51   M         5151515151    Delhi                             12-02-20            P         -
396      908765432199  Rani                         21   F         9977889900    Mumbai                            21-02-20            P         -
1157     909878986789  Anuj                         20   M         9021212121    Bihar                             12-02-20            P         -
1031     979797979797  Rashmika                     32   F         9900099000    Bangalore                         12-02-21            P         -
577      987654321123  Rajesh                       21   M         9900990099    Hyderabad                         20-02-21            P         -
750      987789987789  Ramesh                       32   M         9977553311    Belandur                          18-09-21            P         -
809      990099009900  Ashish                       21   M         8877665544    Bihar                             19-02-21            P         -
690      999888777666  Rakesh                       45   M         9900990099    Hyderabad                         12-02-21            P         -
865      999999888888  Raj                          43   M         9988009988    Kanpur                            19-03-21            P         -


enter value 0 to go to menu
█
```

Figure 4.4 Display Positive Operation

4.4.5 Search Operation

The Figure 4.3 snapshot is of the Search operation where a particular patient record is searched in the B-Tree using Aadhar Number as key.

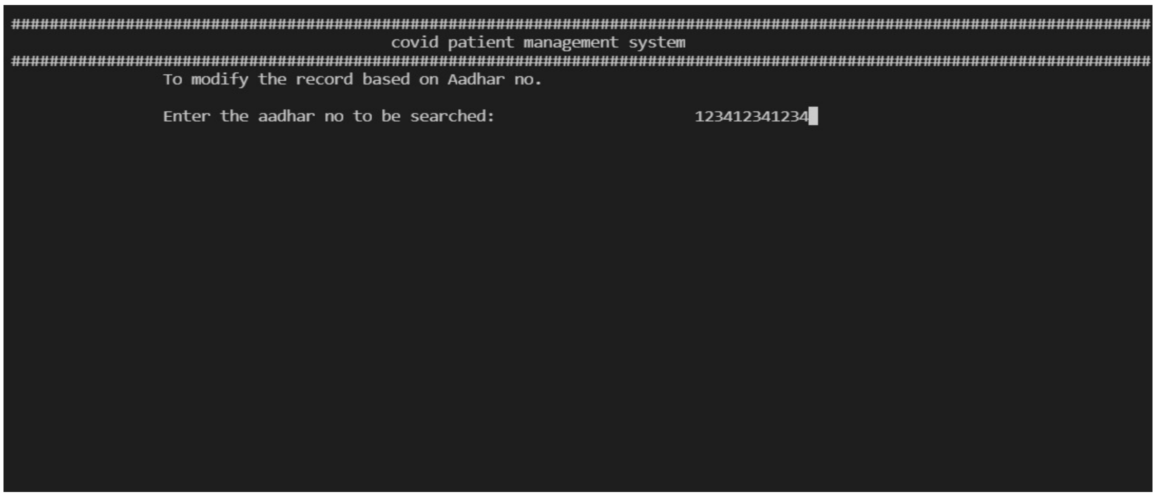


```
##########################################################################################################
                                        covid patient management system
##########################################################################################################
                                        Searching based on Aadhar number


                Enter the aadhar no to be searched:                     123412341234


                adhar No.:123412341234

                Name: Kayak V Gornale

                Age: 21

                Gender: M

                Phone no.: 5454545454

                City: Bangalore

                Date of positive: 12-09-20

                Result: N

                Date of Negative: 10-02-21
                enter value 0 to go to menu:
```

Figure 4.5 Search Operation

4.4.6 Modify Operation

The modify operation is used to search the existing patient record and updates the B-Tree and as well create a new updated record in the Data file.

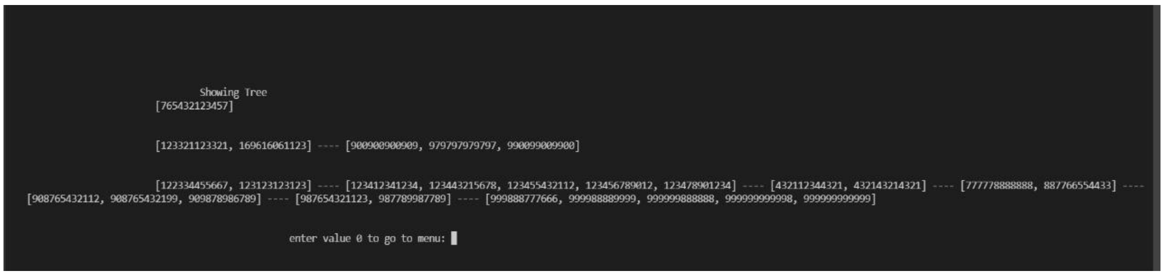


```
##########################################################################################################
                                        covid patient management system
##########################################################################################################
                To modify the record based on Aadhar no.

                Enter the aadhar no to be searched:                     123412341234
```

Figure 4.6 Modify Operation

4.4.7 B-Tree Structure

This option shows us the structure of the B-Tree with Aadhar number as key values.

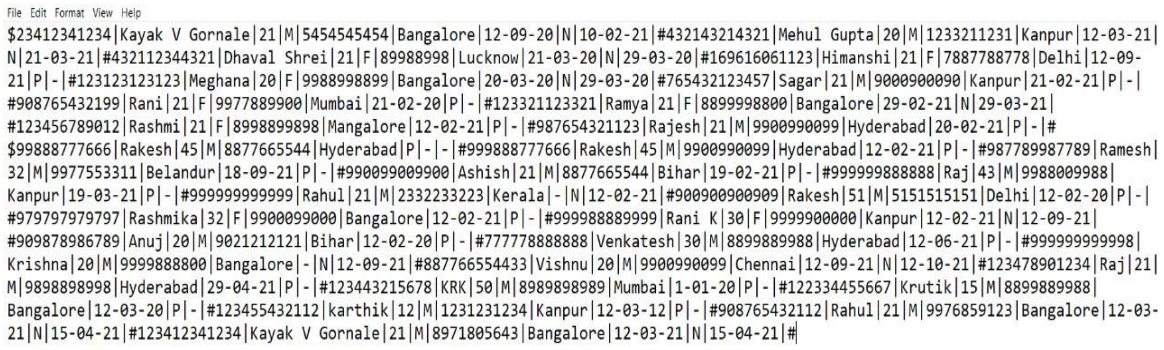


```
                    Showing Tree
               [765432123457]

               [123321123321, 169616061123] ---- [900900900909, 979797979797, 990099009900]

               [122334455667, 123123123123] ---- [123412341234, 123443215678, 123455432112, 123456789012, 123478901234] ---- [432112344321, 432143214321] ---- [777778888888, 887766554433] ----
[908765432112, 908765432199, 909878986789] ---- [987654321123, 987789987789] ---- [999888777666, 999988889999, 999999888888, 999999999998, 999999999999]

                              enter value 0 to go to menu:
```

Figure 4.7 B-Tree structure

4.4.8 File Contents

Here we display the contents of our file which stores all the records separated by a delimiter which is shown in the below figure 4.7.



```
$23412341234|Kayak V Gornale|21|M|5454545454|Bangalore|12-09-20|N|10-02-21|#432143214321|Mehul Gupta|20|M|1233211231|Kanpur|12-03-21|
N|21-03-21|#432112344321|Dhaval Shrei|21|F|89988998|Lucknow|21-03-20|N|29-03-20|#169616061123|Himanshi|21|F|7887788778|Delhi|12-09-
21|P|-|#123123123123|Meghana|20|F|9988998899|Bangalore|20-03-20|N|29-03-20|#765432123457|Sagar|21|M|9000900090|Kanpur|21-02-21|P|-|
#908765432199|Rani|21|F|9977889900|Mumbai|21-02-20|P|-|#123321123321|Ramya|21|F|8899998800|Bangalore|29-02-21|N|29-03-21|
#123456789012|Rashmi|21|F|8998899898|Mangalore|12-02-21|P|-|#987654321123|Rajesh|21|M|9900990099|Hyderabad|20-02-21|P|-|#
$99888777666|Rakesh|45|M|8877665544|Hyderabad|P|-|-|#999888777666|Rakesh|45|M|9900990099|Hyderabad|12-02-21|P|-|#987789987789|Ramesh|
32|M|9977553311|Belandur|18-09-21|P|-|#990099009900|Ashish|21|M|8877665544|Bihar|19-02-21|P|-|#999999888888|Raj|43|M|9988009988|
Kanpur|19-03-21|P|-|#999999999999|Rahul|21|M|2332233223|Kerala|-|N|12-02-21|#900900900909|Rakesh|51|M|5151515151|Delhi|12-02-20|P|-|
#979797979797|Rashmika|32|F|9900099000|Bangalore|12-02-21|P|-|#999988889999|Rani K|30|F|9999900000|Kanpur|12-02-21|N|12-09-21|
#909878986789|Anuj|20|M|9021212121|Bihar|12-02-20|P|-|#777778888888|Venkatesh|30|M|8899889988|Hyderabad|12-06-21|P|-|#999999999998|
Krishna|20|M|9999888800|Bangalore|-|N|12-09-21|#887766554433|Vishnu|20|M|9900990099|Chennai|12-09-21|N|12-10-21|#123478901234|Raj|21|
M|9898898998|Hyderabad|29-04-21|P|-|#123443215678|KRK|50|M|8989898989|Mumbai|1-01-20|P|-|#122334455667|Krutik|15|M|8899889988|
Bangalore|12-03-20|P|-|#123455432112|karthik|12|M|1231231234|Kanpur|12-03-12|P|-|#908765432112|Rahul|21|M|9976859123|Bangalore|12-03-
21|N|15-04-21|#123412341234|Kayak V Gornale|21|M|8971805643|Bangalore|12-03-21|N|15-04-21|#
```

Figure 4.8 File Contents

Chapter 5

CONCLUSIONS AND FUTURE ENHANCEMENTS

- The Covid Patient Record System stores the information of every patient who got tested for covid-19 and its result.
- This software allows storing the details of all the data related to patients.
- Since there can be huge number of the patient records, we use B-Tree indexing for efficient data retrieval, update and storage operations.
- Another advantage of using a B-Tree structure is that is facilitates direct access of records.

Future Enhancements are as follows:

- We can add a functionality where we can retrieve the patient records which were tested positive and needs to be re-tested after 14 days.
- We can add functionality where we get the option to filter out the records based on certain parameters such as city, gender etc.
- We can implement Hashing which allows record access in only one or very few runs with the help of a hashing function.

REFERENCES

[1] Michael J Folk, Bill Zoellick, Greg Riccardi: File Structures – An Object Oriented Approach with C++, 3rd Edition, Pearson Education, 1998

[2] K.R. Venugopal, K.G. Srinivas, P.M. Krishnaraj: File Structures Using C++, Tata McGraw-Hill, 2008.

[3] https://www.cplusplus.com/

[4] www.geeksforgeeks.com

[5] www.youtube.com (Abdul Bari Channel)