**CHAPTER 1**

**INTRODUCTION**

Aadhaar is here to stay for the long run, although there are still a few kinks that the centre need to work out. The objective of the scheme has been to issue a unique identification number by the Unique Identification Authority of India, which can be authenticated and verified online. It was conceptualized and implemented as a platform to facilitate identification and avoid fake identity issues

**1. PROBLEM DEFINITION**

At a certain point it became difficult for the government to manage the details about each person in the country. The Aadhar card was launched by the government so a person can have a single unique identification document or number that would capture all the details of that person including biometric and demographic information. Many cases, where sensitive Aadhaar identity information was leaked and made public. There have also been instances where bank accounts linked with Aadhaar were vulnerable to hacking. This project relates to the real-world problem in providing a secure Aadhaar card. A report from a year ago implied that several parties illegally tried to store the biometric data and conduct multiple transactions using the same fingerprint. As some of these problems with Aadhaar makes it less-efficient than it actually is.

**2. OBJECTIVES**

This project mainly concentrates on increasing the efficiency of Aadhaar card by searching and removing the fake Aadhaar application which are misused for various activities. An Aadhaar is created and stored in the tree only when the necessary and legal information are submitted by the person. The people residing to different places come under different list. So, people are grouped based on their state of residence. A unique identification number is provided for the person. By this UID number it becomes easier to search for any duplicates and to maintain in database (in this project details are stored in nodes of tree). By this there is transparency of all the details stored. Recently Blockchain Technology has taken its way to improvise Aadhaar in terms of security and privacy. It would allow information to be collected, held and utilized transparently with the consent of the individual whose information it is. While using blockchain for Aadhaar, there will be multiple trusted nodes. Trusted nodes will be able to validate a transaction and append blocks in the blockchain. Only these trusted nodes will be able to decrypt the data stored in the blocks.

**3. METHODOLOGY TO BE FOLLOWED**

This mini project makes use of the following data structure to meet the needs of the following objectives:

* The process of creation and insertion of the new candidates’ details is done inside a node of a tree structure. Each node is assigned for one candidate.
* The process of displaying the entire candidates details of that tree is done using the display function.
* The process of searching the details of a particular candidate is using the search function.
* Finally, the feedbacks from the candidates are take and stored in a file using file concepts.

**4. EXPECTED OUTCOMES**

There are a variety of end uses of Aadhaar card. It is pretty much necessary for a basic civic life in the country. This mini project aims to give the following outcomes:

* The main objective is to provide unique identification card to all the candidates who are registering to create an Aadhaar account.
* It helps in avoiding duplication of true identity of a person. One Aadhaar number for one person.
* It helps in keeping a count on the population of a state an indirectly even the country(census).
* Reduces the cost of identifying persons and provides increased transparency of all the details stored.

**5. HARDWARE AND SOFTWARE REQUIREMENTS**

**Hardware Requirements:**

* RAM: 4GB and above
* Device Name: DELL Vostro 5481 laptop

**Software Requirements:**

* Operating System: Microsoft Windows 8 and above
* Processor: Processor Intel(R) Core i5-8265 8th Gen
* Language: C
* Compiler: Dev C++ or Code: blocks or Turbo C++

**CHAPTER 2**

**DATA STRUCTURES**

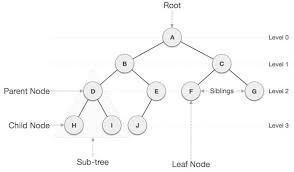
**1. TREE**

A tree is a non -linear data structure and a widely used abstract data type (ADT). It simulates a hierarchical tree structure, with a root value and subtrees of children with a parent node, represented as a set of linked nodes.

A tree data structure can be defined recursively as a collection of nodes (starting at a root node), where each node is a data structure consisting of a value, together with a list of references to nodes (the "children"), with the constraints that no reference is duplicated, and none points to the root.

**Binary Search Tree** is a node-based binary tree data structure which has the following properties (the conditions for a tree to be binary):

* The left subtree of the root node contains only nodes with keys lesser value of the root nodes key.
* The right subtree of the root node contains only nodes with keys greater value of the root nodes key.
* The left and right subtree each must also be a binary search tree.



**Fig. 2.1** Structure of a Binary Search Tree

**1.1 TERMINOLOGIES USED IN A TREE ARE:**

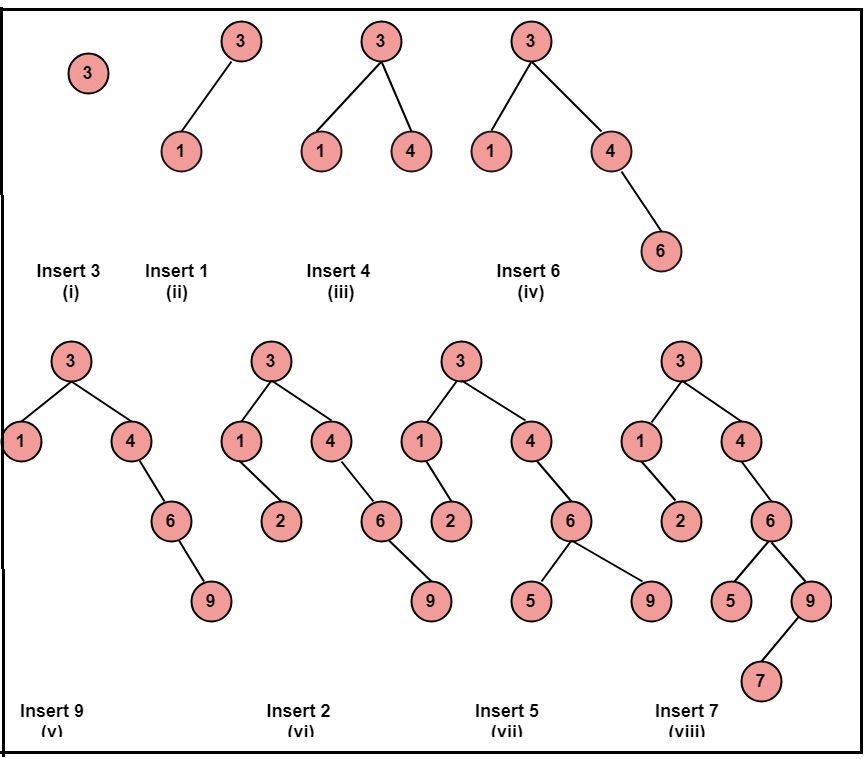
* **Root**: In a tree data structure, the first node is called as Root Node. Every tree must have a root node. We can say that the root node is the origin of the tree data structure. In any tree, there must be only one root node. We never have multiple root nodes in a tree.
* **Edge**: In a tree data structure, the connecting link between any two nodes is called as EDGE. In a tree with 'N' number of nodes there will be a maximum of 'N-1' number of edges.
* **Parent**: In a tree data structure, the node which is a predecessor of any node is called as PARENT NODE. In simple words, the node which has a branch from it to any other node is called a parent node.
* **Child**: In a tree data structure, the node which is descendant of any node is called as CHILD Node. In simple words, the node which has a link from its parent node is called as child node. In a tree, any parent node can have any number of child nodes. In a tree, all the nodes except root are child nodes.
* **Siblings**: In a tree data structure, nodes which belong to same Parent are called as SIBLINGS. In simple words, the nodes with the same parent are called Sibling nodes.
* **Leaf:** In a tree data structure, the node which does not have a child is called as LEAF Node. In simple words, a leaf is a node with no child.

# Advantages of a binary search tree over a linked list:

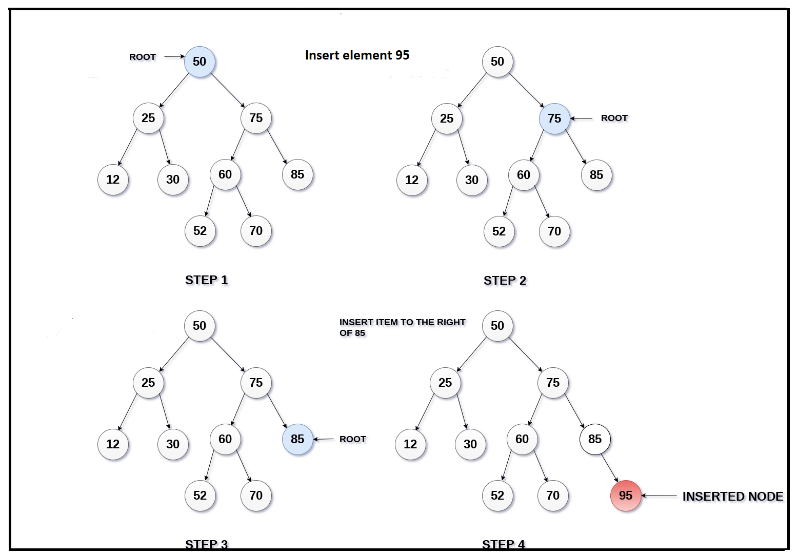
* Binary search tree has efficient search function (i.e. in order to find a specific element you don't have to look at all the elements).
* LinkedList is an O(N) traversal data structure, while a BST is an O(N) traversal data structure in the worst case, and a O (log N) in the best case.
* Search - search is really quick as complexity is O(h)O(h) ***h***is the height of the tree. On the other hand, search in linked list is O(n)O(n).
* If you have a large amount of fairly static data which you need to search a lot, and you have some kind of ordering relationship, then building a balanced tree out of it might be worth doing instead of using a linked list.
* When you insert into a binary search tree, where those elements end up being stored in memory is a function of their value. With a linked list, elements are blindly added to the list regardless of their value.
* Trees are hierarchical way of storing data which is organized according to their data value. Whereas in linked list it is inserted at random places and if we have to insert the data in a sequential in is time consuming.

**1.2 OPERATIONS ON A TREE:**

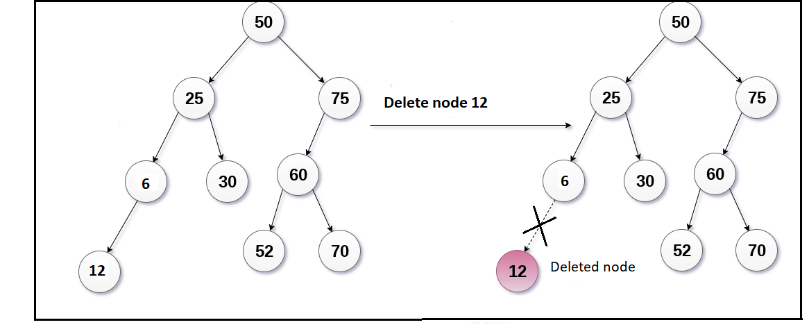
* CREATE and INSERT: Binary tree is created by inserting root node and its child nodes. The first element inserted is considered as the root node. The next consecutive elements entered, depending on the value of the element the element is inserted to the left sub tree if its value is less than root else it is inserted into the right sub tree.
* SEARCH: Searching is done as per value of node to be searched whether it is root node or it lies in left sub tree or right sub-tree.
* DELETE: Binary tree is deleted by removing its child nodes and root node.



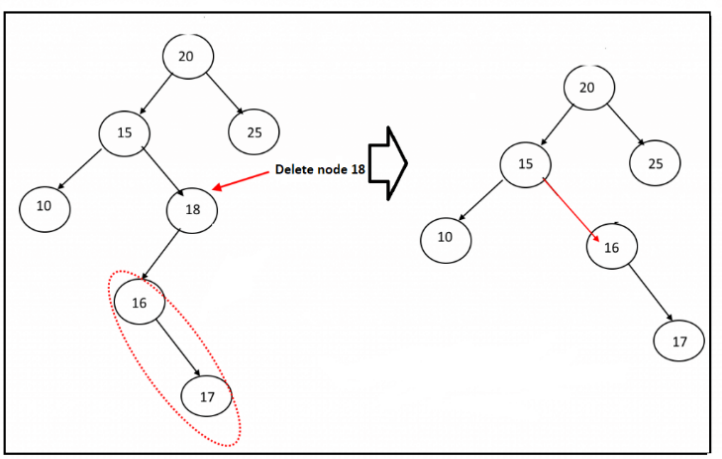
**Fig.2.2.1** Example of creating and inserting in a BST.



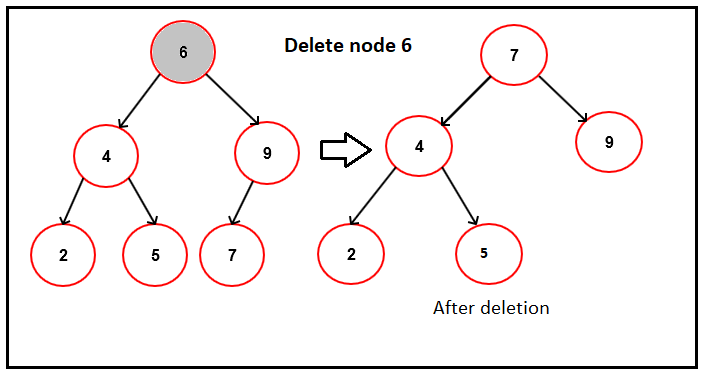
**Fig.2.2.2** Insertion of a node



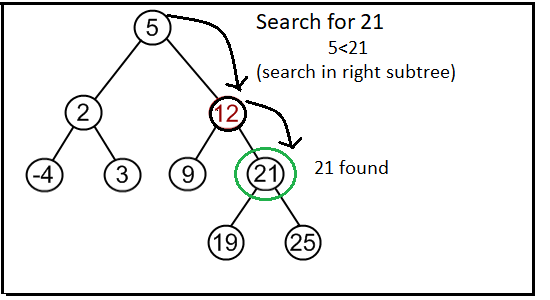
**Fig.2.2.3** Deletion of a leaf node



**Fig.2.2.4** Deletion of a node with one child



**Fig.2.2.5** Deletion of a node with two children



**Fig.2.2.6** Search for a node in the tree

**CHAPTER 3**

**DESIGN**

**1.DESIGN GOAL:**

In the process of creating Aadhaar card we are using the concept of Trees to populate the information of a candidate in terms of Legal Name, DOB, Gender, Father’s Name, Address, Age, gender. At the same time each individual is given a unique ID called UID number allotted to them. Then all these details are stored in each nodes of a tree respective of their state. This process continues for several individual.

**2. ALGORITHM/PSEUDO CODE:**

**Pseudo code to:**

**1.1 Create a tree and populate the details of the candidate in the node:**

Step 1: START

Step 2: Declare a structure with arguments as name, DOB, age, fathers name, door

number, gender, address as the data part. ‘l’ and ‘r’ be the pointers to left and right

child nodes.

Step 3: declare \*root=NULL, \*cur=NULL, \*t2, \*t1

Step 4: For inserting the first candidates’ details we create a node by void insert ()

Print: Enter your name, age, father’s name, address.

Scan the details and dynamically allocate memory for cur

cur->value = name

cur->value1 = age

cur->value2 = DOB

cur->value3 = fathername

cur->value4 = gender

cur->value5 = address

cur->l = cur->r = NULL

Step 5: Search for the right location to insert the next node

void insert (node \*cur, node \*t)

// value less than root node, insert value at right sub tree

if cur->value > t->value1 && t->r! = NULL

insert (cur->r, t)

else if cur->value > t->value1 && t->r == NULL

t->r = cur

// value less than root node, insert value at left sub tree

else if temp->value < t->value1 && t->l! = NULL

insert (cur->l, t)

else if temp->value < t->value1 && t->l == NULL

t->l = temp

Step 6: END

**1.2 Generating the UID for each candidate:**

Random (int g, int h, int v)

int i, num;

for (i = 0; i < v; i++)

num = (rand () % (g - h + v)) + g

print: num

Now scan this UID into cur->uid

**1.3 Display all the candidate’s details:**

Step 1: START

Step 2: Display the entire list INORDER format

void inorder (struct node \*cur)

Step 3: if root == NULL

Print that no candidates have registered

Return

Step 4: If (cur->l! = NULL)

inorder(cur->l)

print cur->value = name

cur->value1 = age

cur->value2 = DOB

cur->value3 = fathername

cur->value4 = gender

cur->value5 = address

cur->value6 = UID

Step 5: if t->r! = NULL

inorder (cur->r)

Step 6: END

**1.4 Delete a particular candidate from the tree:**

Step 1: START

Step 2: Read the age and the UID of the candidate. Declare the pointers parent= NULL

and child. Initialise the pointer cur=root;

Step 3: void delete (node\* root, int uid)

while (cur not equal to NULL and cur->n not equal to uid)

parent=cur;

If (uid is less than cur->data)

cur=cur->lch

else

cur=cur->rch

if (cur is equal to NULL)

print Your Aadhaar account was not found

Step 4: //Deleting a leaf node

if(cur->lch is equal to NULL and cur->rch is equal to NULL)

if (cur not equal to root)

if (parent->lch equal to cur)

parent->lch=NULL

else

parent->rch=NULL

else

root equal to NULL

delete cur

Step 5: //Deleting a node with two children

else if((cur->lch not equal to NULL) and(cur->rch not

equal to NULL))

node \*successor=min(cur->rch)

int val=successor->data

del (root, successor->data)

cur->data=val

Step 6: //Deleting a node with one child

else if((cur->lch not equal to NULL) and(cur->rch equal

to NULL))

if(parent->lch==cur)

parent->lch=cur->lch

else

parent->rch=cur->lch

delete cur

else if((cur->lch equal to NULL) and (cur->rch not equal to NULL))

if(parent->lch==cur)

parent->lch=cur->rch

else

parent->rch=cur->rch

delete cur

Step 7: END

**1.5 Search for a particular candidate details in the tree:**

Step 1: START

Step 2: Ask the user to enter age and UID number as per mentioned in Aadhaar card

void search (a\* cur, int age, int id)

Step 3: while (cur not equal to NULL and uid not equal to cur->uid)

if (pid is less than cur->n)

cur = cur->l

else

cur = cur->r

Step 4: if (cur is equal to NULL)

print: Your details not found

else

print: “Yes, your AADHAAR account exist. Your details as per our database is: "

print: cur->value = name

cur->value1 = age

cur->value2 = DOB

cur->value3 = fathername

cur->value4 = gender

cur->value5 = address

cur->value6 = UID

Step 5: END

**1.6 Feedbacks from candidates:**

Step 1: START

Step 2: Open a file in write mode and append mode and in .txt format.

Declare a file pointer as FILE \*fp

Step 3: print You can write down your feedback here

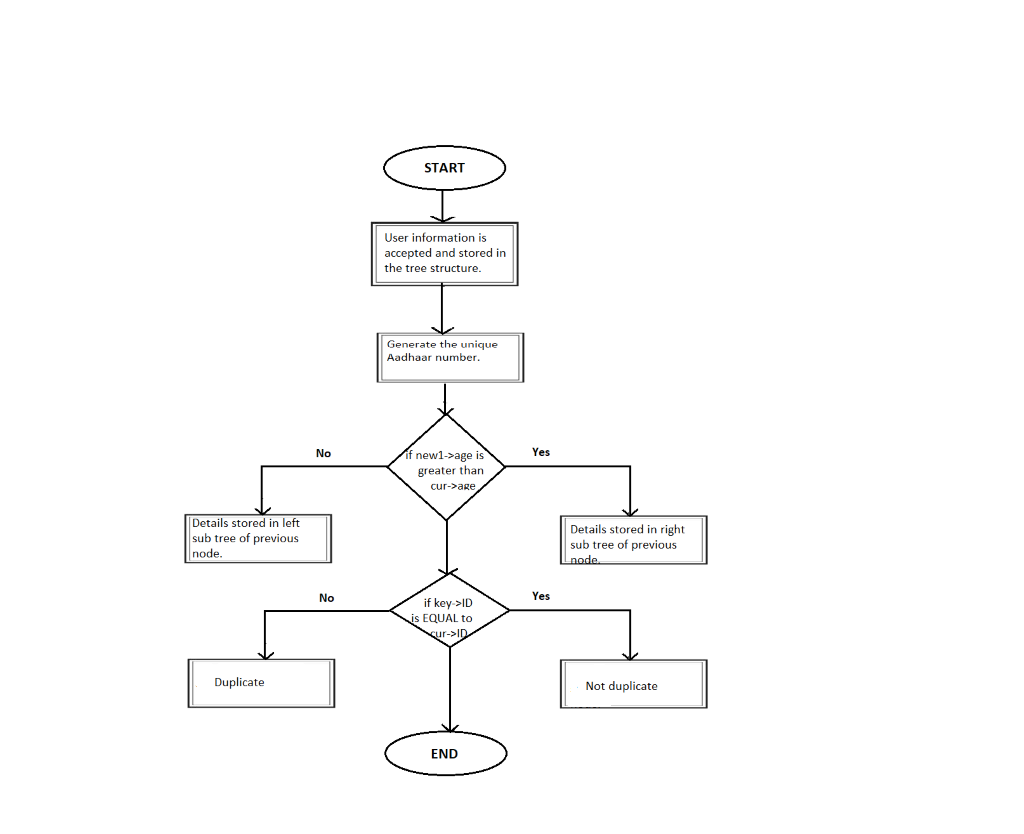
Step 4: while ((ch=getche ()) not equal to '\t')

Put every character into the file using the file pointer fp.

Step 5: At last close the file.

Step 6: END

**3. FLOWCHART**



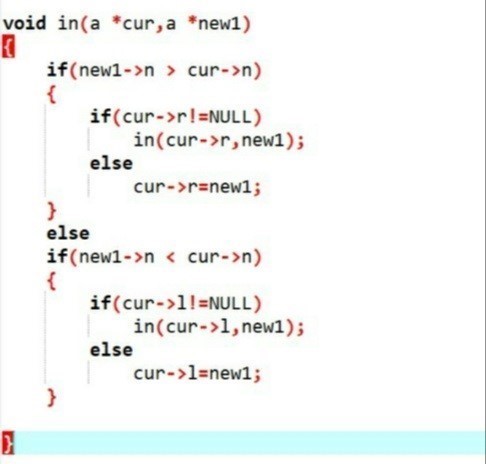
**Fig.3.1** Flowchart for Aadhaar Card Generation

**CHAPTER 4**

**IMPLEMENTATION**

**1.IMPLEMENTATION OF DATA STRUCTURES**

**Module 1 Functionality:**

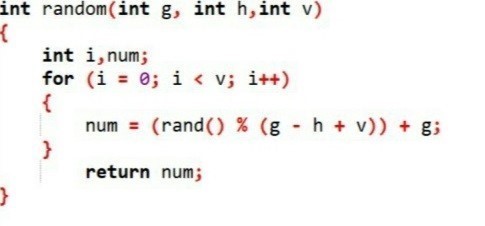


**Fig.4.1** Insert function

This module deals with the process of creating and inserting the details of the candidates using insert (). Insertion is done with respect to the age of the candidate. The first candidate’s details are stored in the root node of the tree corresponding to the state selected. Then the next candidates age is compared to root->age, if greater it gets inserted into the right sub tree else into the left sub tree.

Pointers root1, root2, root3, root4, root5 are declared for the states Karnataka, Tamil Nadu, Kerala, Andhra Pradesh and Goa respectively. The details of the candidates are populated in the tree with respect to the state they belong to.

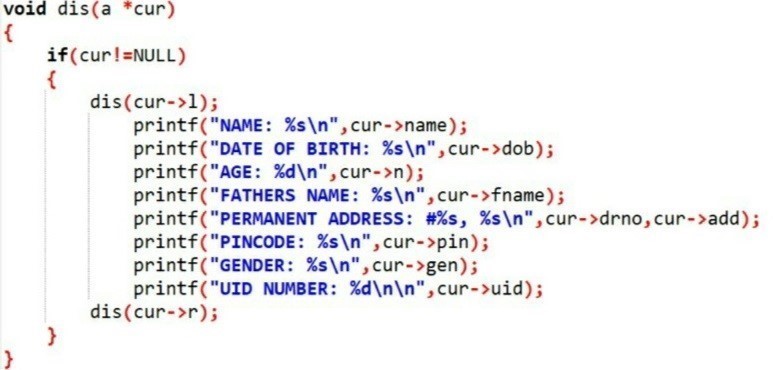
**Module 2 Functionality:**



**Fig.4.2** Function to generate UID number

This module relates to the generation of the UID number. Every candidate after providing the required details is allotted with a Unique Identification Number. The process of generating this number is with the help of random (). The srand function uses the time (0) to get the current time and calculates the above expression and returns a number. This number is given for the candidate as his/her UID number.

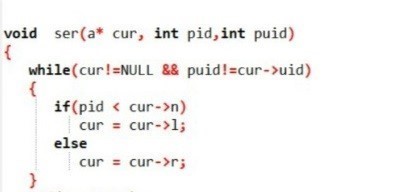
**Module 3 Functionality:**



**Fig.4.3** Function todisplay all the registered candidates’ details

This module shows us the process of displaying every candidates detail, we use the INORDER traversal method, in which the left node is printed first then the root and the right node at last.

**Module 4 Functionality:**

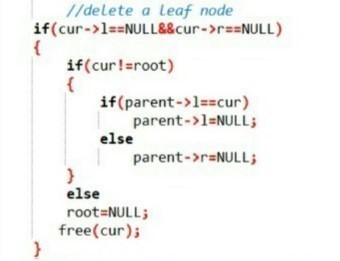


**Fig.4.4** Function to search for a particular candidate

This module explains the process of searching a particular candidate from the tree structure. We ask for the candidates age and UID number. With the help of age, we get to know if the candidate’s detail is stored in the right or left sub tree. Then by comparing the UID number with each node we can display the candidate’s details if present else we print details were not found.

**Module 5 Functionality:**

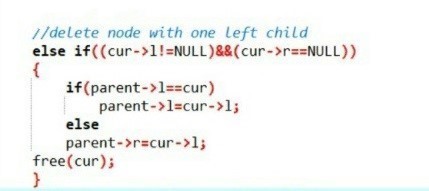
There are 3 cases in which a node can be deleted.



**Fig.4.5.1** Function to delete a leaf node

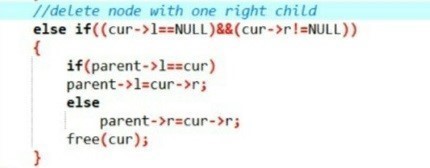
A node which does not have either left or right child is called a leaf node.

The pointer traverses to last node in which the UID matches and deletes that node.



**Fig.4.5.2** Function to delete a node with one left child

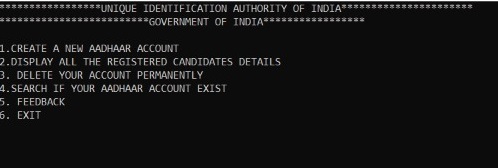
* The condition of a node where its right pointer is NULL and left pointer not NULL is used to delete the left node. It is a recursive function in order to traverse to the required node.



**Fig.4.5.3** Function to delete a node with one right child

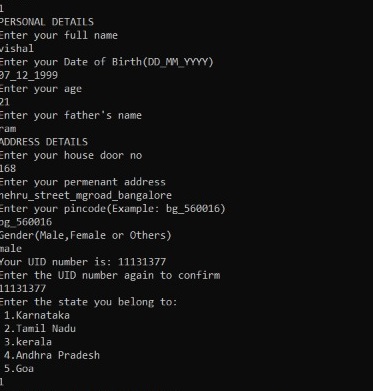
* The condition of a node where its right pointer is NULL and left pointer not NULL is used to delete the left node. It is a recursive function in order to traverse to the required node.

**5.RESULTS:**

****

**Fig.5.1** User – friendly menu

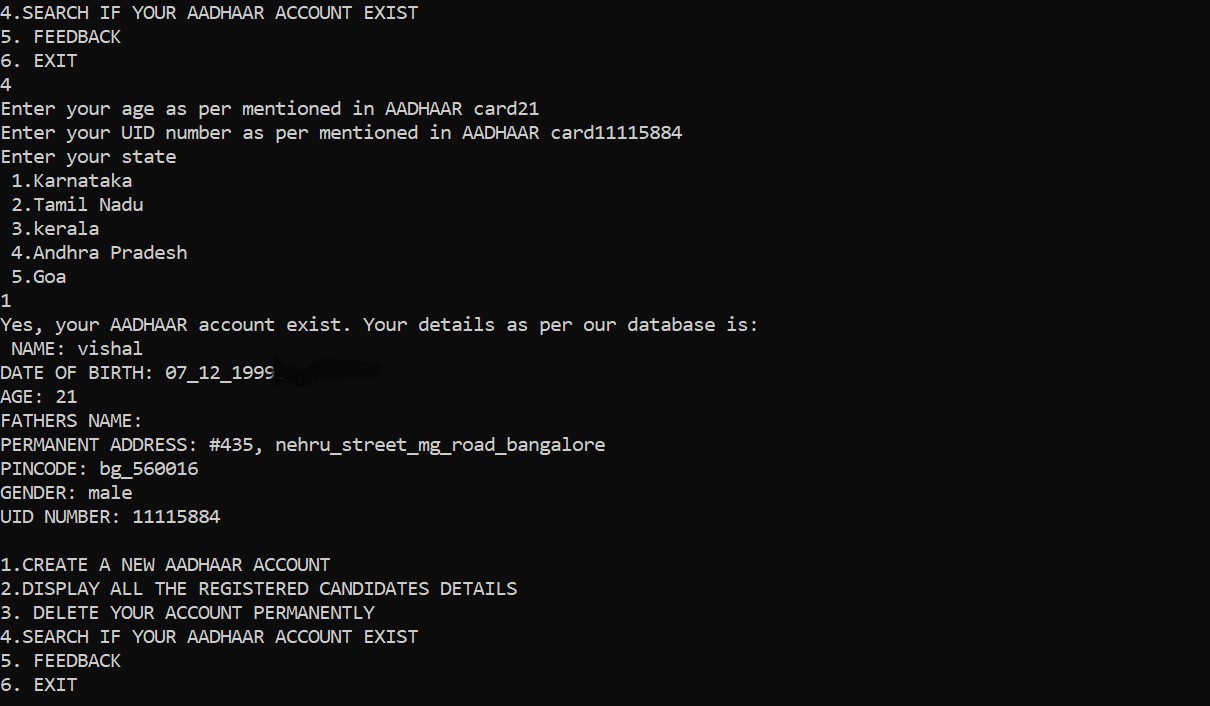
The final output shows the various options like create, display, delete, search, feedback and exit. The user chooses option “1” if he has to create a new Aadhar account.



**Fig.5.2** Details to create an Aadhaar card

On creation of Aadhaar account details like name, age, DOB, father name, address, gender and finally the state he/she belongs to.

An Aadhar number is generated and provided it to the candidate at this stage and the details are stored in the tree. This number is later used to search or delete a particular node.



**Fig.5.3** Search a particular candidate

In order to search for a particular candidate, we ask for their age and UID number. With the help of age, we get to know if the candidate’s detail is stored in the right or left sub tree. Then by comparing the UID number with each node we can display the candidate’s details if present else we print details were not found.

When the user wishes to delete their Aadhaar account then same as search function we ask for their age and UID number. With the help of age, we get to know the location of the candidate’s detail if it’s on the right or left sub tree. in the tree.

This project simulates in creating and providing Aadhaar card in an organised and user – friendly manner. This project takes the help of C programming language to build the code.**C program is useful for high level programming language and very efficient for general purposes, it offers performance and memory efficiently.**

**Hence this project is able to accomplish the process of generating Aadhaar card (UID number) and explaining the add on features to it. The usage of Data structures like trees, linked list, queues, stacks provides a way to organize data by which the data can be accessed easily later when required.**

**Therefore, C programming language provide us several platforms to create real time projects.**

**6.CONCLUSION:**

There are various future scopes of Aadhaar card. This project can take on many more add on features like Aadhaar authentication, in which the resident can authorise to release their details to service providers, either in person or online. Only on successful authentication the details of the resident like photo, address, date of birth, gender will be sent to the service providers. By doing this it turns out to be a profit for both the resident (the candidate) and the service providers.

In the similar way even biometric capture technology can be implemented which makes it further more easier to get the details of a person**. Therefore, this project relates to a real-world problem and provides a solution for it.**

**7. REFERENCES:**

The following websites are referred to create this project:

1. [https://www.slideshare.net /aadhaar-card-registration-system?next\_slideshow=1](https://www.slideshare.net/PriyankaSingh958/aadhaar-card-registration-system?next_slideshow=1)

2.<https://www.tutorialspoint.com/data_structures_algorithms/tree_data_structure.htm>

3. <https://www.sanfoundry.com/c-program-construct-binary-search-tree/>

4. <http://aircconline.com/ijnsa/V9N3/9317ijnsa01.pdf>

5.<https://www.tutorialspoint.com/data_structures_algorithms/tree_data_structure.htm>

6. <https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/>