

PHONE BOOK DIRECTORY USING DATA STRUCTURES

by

Hari Krishna	20BEC1004
Karthikeyan S	20BEC1241
Harish D	20BEC1034
Gabriel K	20BEC1320

A project report submitted to

Dr .E. MANIKANDAN

SCHOOL OF ELECTRONICS ENGINEERING

in partial fulfilment of the requirements for the course of

CSE2003 – Data Structures and Algorithm

in

B.Tech. Electronics and Communication Engineering



VIT[®]

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

VIT CHENNAI

Vandalur-Kelambakkam Road

Chennai – 600127

APRIL 2022

BONAFIDE CERTIFICATE

Certified that this project report entitled “**PHONE BOOK DIRECTORY USING DATA STRUCTURES**” is a Bonafide work of **Hari Krishna, Karthikeyan S, Harish D, Gabriel K** who carried out the project work under my supervision and guidance.

Dr.E.Manikandan

Assistant Professor Senior

School of Electronics Engineering (SENSE),

VIT Chennai

Chennai-600127.

ABSTRACT

This project (Phonebook directory) supports searching and sorting using just a few characters of the search criteria. We will be able to search by phone number first-name, last-name, or even a portion of a name. Search results can return one or multiple items and the list comes back in sorted order. We will implement such a system that the phone data will be stored in a Ternary search tree using hash table.

Keywords: TST, Hashing, Hash table.

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr.E.Manikandan**, Associate Professor Senior, School of Electronics Engineering, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

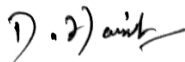
We are extremely grateful to **Dr. A. Sivasubramanian**, Dean of the School of Electronics Engineering, VIT Chennai, for extending the facilities of the school towards our project and for her unstinting support.

We also take this opportunity to thank all the faculty of the school for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.



HARI KRISHNA S



HARISH D



GABRIEL K



KARTHIKEYAN S

TABLE OF CONTENTS

S.NO	TOPIC	PAGE NO.
1	INTRODUCTION	6
2	OBJECTIVE AND GOALS	6
3	BENEFITS	6
4	PROBLEM STATEMENT	7
5	JUSTIFICATION OF DATA STRUCTURE	7
6	DESIGN OF DATA STRUCTURE	8
7	TST ALGORITHM	9
8	PROGRAM CODE	11
9	OUTPUT	36
10	LIMITATIONS	39
11	APPLICATIONS AND ADVANTAGES	39
12	CONCLUSION	40
13	REFERENCES	40
14	ABOUT US	41

1.INTRODUCTION

1 OBJECTIVES AND GOALS

- Design TST Algorithm.
- To design an algorithm to perform search and modify operations in the phonebook dataset
- Save the changes and load the modified dataset while performed again.

2 BENEFITS

- The advantage of using ternary search trees over tries is that ternary search trees are more space-efficient (involve only three-pointers per node as compared to 26 in standard tries).
- Further, ternary search trees can be used any time a hash table would be used to store strings.
- Ternary search trees are efficient to use (in terms of space) when the strings to be stored share a common prefix

3 FEATURES

- It will be an easy process for the users to maintain and manage their contacts list and their details
- Created a folder for this particular phonebook and under that 26 text dictionary files each for each alphabet for saving and retrieving
- Ternary search trees are preferable because we may have the strings to be stored all have the same prefix

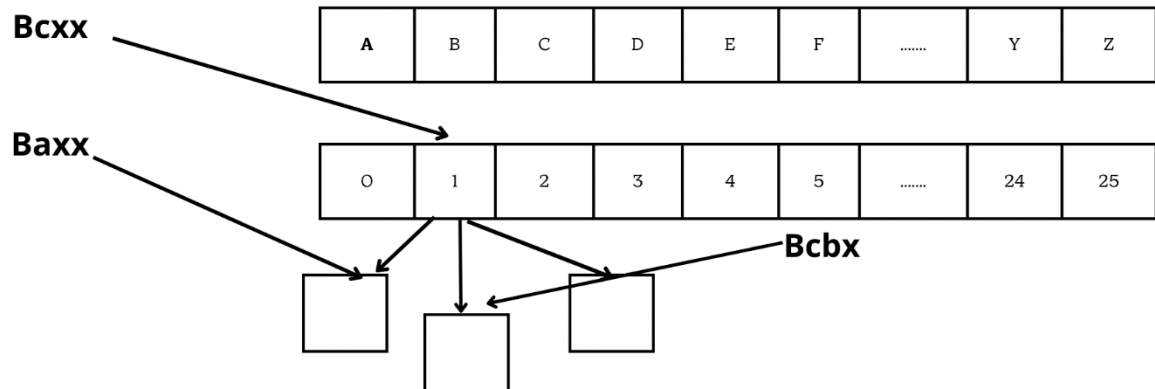
4 PROBLEM STATEMENT

- To develop a phone directory system that stores the subscriber information in the order of subscription and that can
 - a. store
 - b. delete
 - c. modify &
 - d. perform other operations on the large number of data effectively and accommodate the data complexity in real world model.
- We can implement TST (Ternary Search Tree) algorithm to solve this real-life application.
- Personal information such as name, phone number, email and address are asked while adding a record into the Phonebook.
- These records can then be modified, listed, searched for and removed using our project

5 JUSTIFICATION OF DATA STRUCTURE

- A ternary search tree is a special tree data structure where the child nodes of a standard tree are ordered as a binary search tree
- Unlike other search tree, here each node contains **26 pointers** for its children, each node in a ternary search tree contains only **3 pointers**
 1. The **left pointer** node whose value **is less** than the value in the current node.
 2. The **equal pointer** node whose value **is equal** to the value in the current node.
 3. The **right pointer** node whose value **is greater** than the value in the current node.
- Our project could also be done with binary search tree as it also has some advantages over ternary search tree, but when we insert two data set with same parent node(name) we cannot store the second node properly
- So, thereby using the ternary search tree algorithm we sorted out the problem
- Here, when the nodes are equal the data will be pointed towards the equal (third) node so that there are no discrepancies

6 DESIGN OF DATA STRUCTURE



We created a folder for this particular phonebook and under that 26 text dictionary files each for each alphabet (For example: Directory_A.txt, ..., Directory_Z.txt)

For retrieving the data from each text file we retrieved with its path and using a for loop so that every text file is loaded

LOAD: Our files will get loaded each time, when we compile and run our program

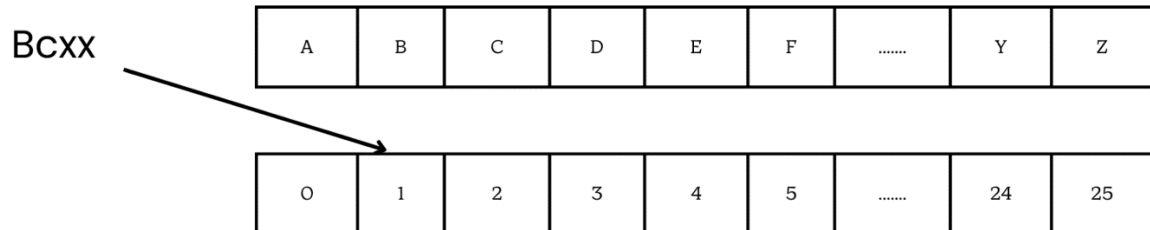
SAVE: Similarly, our data will be saved every time while we give the EXIT as a choice after performing the user required operations to the phonebook

7 TST ALGORITHM

A ternary search tree is a special tree data structure where the child nodes of a standard tree are ordered as a binary search tree.

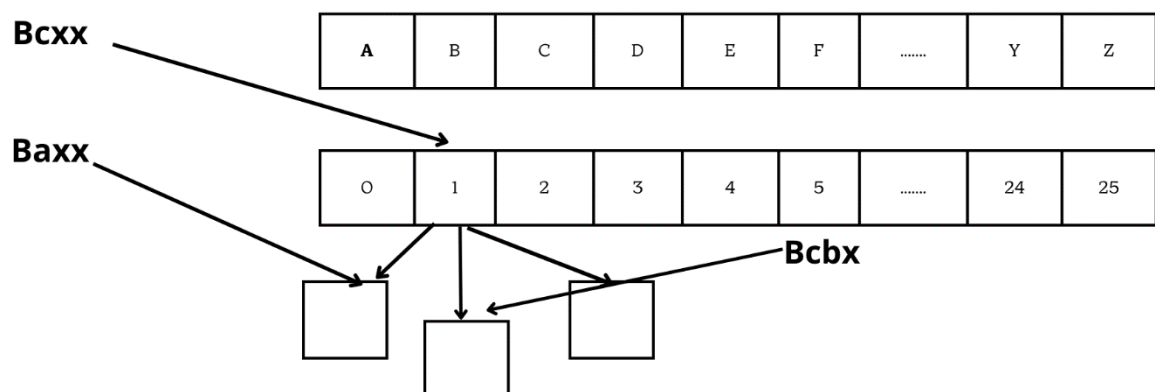
INSERT OPERATION

Let Name: BCXX



$$\text{hash} = \text{int}(\text{name}[0]) \% 65$$

- While inserting we get the ASCII value of first letter of the name and perform hash operation so that the data is arranged in alphabetical order. Following this BCXX is added to array index 1
- The hash algorithm discussed above is prone to collisions when data entries duplication each other (when names are same or the first letter matches).
- In this case, collision handling approach is based on separate chaining but with not with a linked list but a tree.
- Thus, the hash array holds chain of trees and each tree branch demonstrates the subscribers with root being the first among the subscription.



- When a subscriber with similar name has to entry their details, the true value of ternary search tree can be recognized.
- The ternary search tree (TST) can compare similar values and if, that is the case then, the respective value will be placed in the middle.

SEARCH AND MODIFY

- Get the name and number from the user which is to be searched
- Return the pointer of the particular node
- Find which data to be modified under that particular node
- Modify it and exit

DELETE

- Get the name and phone number from the user which is to be deleted.
- Traverse to the respective node
- Compare the phone number of the node using string compare.
- Check if the **mid-pointer** is empty or not
- If empty, find minimum value from right child, and replace the value of the deleted node.
- Else, delete the node and replace the node with the mid node.

TIME COMPLEXITY:

- The time complexity of the ternary search tree operations is similar to that of binary search tree.
- The insertion, deletion, and search operations take time proportional to the height of the ternary search tree.
- The space is proportional to the length of the string to be stored.

8 PROGRAM CODE

```
#include<iostream>
```

```
#include<stdlib.h>
```

```
#include<string.h>
```

```
#include<fstream>
```

```
#include<conio.h>
```

```
#include<stdbool.h>
```

```
#include <Windows.h>
```

```
using namespace std;
```

```
struct node
```

```
{
```

```
char name[30];
```

```
char number[15];
```

```
char address[40];
```

```
char email[40];
```

```
struct node *left, *right, *mid;
```

```
};
```

```
struct node* searchin(struct node* root,char name[],char number[]);
```

```
void modify(struct node* ptr);
```

```
void savein(struct node *head,char fn[]);
```

```
void gosave();
```

```
void insertstackpush(struct node* temp);
```

```
void load();
```

```

struct node* pop(struct node*stack);

void display();
void print();
void save();

void savein(struct node *head,char fn[]);
struct node *input(struct node *);
void deletestackpush(struct node* );
struct node* deleteNode(struct node*root[], char key[]);
struct node* deleteNodeinside(struct node*, char key[],char number[]);
struct node *findmin(struct node *tree);
void inorder(int i);
void inorderinside(struct node *);

struct node *insertinside(struct node *,struct node *);
struct node *findmin(struct node *tree);
struct node *insertstack = NULL,*deletestack = NULL,*head[26];

int main(){
struct node *n_node,*ptr;
char ans;
char n[30],num[15];
int hash,k =0;
for(int i=0;i<26;i++)
{
head[i]=NULL;

```

```

    }

    int option;

    char key[40],keyn[40];

    int temp_status;

    printf("\n*****
    *****PHONEBOOK*****
    *****
    *****");

    printf("\n-----
    -----
    -----");

    load();

    do{

    cout<<"\nMENU\n1)INSERT\n2)PRINT\n3)SEARCH AND
    MODIFY\n4)DELETE A CONTACT\n5)EXIT\nENTER YOU
    CHOICE\n";

    cin>>option;

    switch(option)

    {

    case 1:

    do{

    n_node = input(n_node);

    hash = int(n_node->name[0]) % 65;

    insertstackpush(n_node);

    head[hash] = insertinside(head[hash],n_node);

    cout<<"\nCONTINUE ? (y/n)\t";

    cin>>ans;

    }while(ans == 'y' || ans == 'Y');

```

break;

case 2:

print();

break;

case 3:

cout<<"ENTER THE SUBSCRIBER'S NAME TO BE SEARCHED\n";

cin.getline(key,40);

while(strlen(key)==0)

cin.getline(key,40);

cout<<"ENTER THE SUBSCRIBER'S NUMBER\n";

cin>>keyn;

while(key[k] != '\0'){

key[k] = toupper(key[k]);

k++;

}

hash = int (key[0]) % 65;

ptr = searchin(head[hash], key,keyn);

if(ptr!=NULL){

cout<<"\n\nNAME : "<< ptr -> name<<endl;

cout<<"NUMBER : "<< ptr -> number<<endl;

```

cout<<"ADDRESS : "<< ptr -> address<<endl;
cout<<"E-MAIL : "<< ptr -> email<<endl;

cout<<"\n\nDo you want to modify? (y/n)\n\n"<<endl;
cin>>ans;
if(ans == 'y' || ans == 'Y'){

    modify(ptr);
}
}

k=0;

break;

case 4:
cout<<"ENTER THE NAME\n";
cin.getline(key,40);

while(strlen(key)==0)
cin.getline(key,40);

cout<<"ENTER THE NUMBER\n";
cin>>num;

while(key[k] != '\0'){
key[k] = toupper(key[k]);
k++;
}

```

```
hash = int (key[0]) % 65;  
head[hash] = deleteNodeinside(head[hash],key,num);  
k=0;
```

```
break;
```

```
case 5:
```

```
break;
```

```
default:
```

```
cout<<"ENTER VALID CHOICE\n";
```

```
break;
```

```
}
```

```
}while(option!=5);
```

```
gosave();
```

```
return 0;
```

```
}
```

```
struct node *insertinside(struct node *head,struct node *new_node)
```

```
{
```

```
if(head == NULL)
```

```
{
```

```
new_node -> left = NULL;
```

```
new_node -> right = NULL;
```

```
new_node->mid = NULL;
```

```
head = new_node;
```



```

    }
    else
    {p

    if(strcmp(head -> name, new_node -> name) == 0)
    {
    if(strcmp(head -> number, new_node -> number) == 0){
    cout<<"\n\t\tAlready exist!!!!"<<endl;
    }
    else{
    head -> mid = insertinside(head -> mid, new_node);

    }

    }
    else if(strcmp(head -> name, new_node -> name) > 0)
    {
    head -> left = insertinside(head -> left, new_node);

    }
    else
    {
    head -> right = insertinside(head -> right, new_node);

    }
    }

    return head;

```

```
}
```

```
void print()
```

```
{
```

```
for(int i =0;i<=26;i++){
```

```
if(head[i] == NULL){ continue;}
```

```
if(head[i] != NULL){
```

```
inorderinside(head[i]);
```

```
}
```

```
}
```

```
cout<<"\n";
```

```
}
```

```
void inorderinside(struct node *head)
```

```
{
```

```
if(head != NULL){
```

```
inorderinside(head -> left);
```

```
cout<<"\n\nNAME : "<< head -> name;
```

```
cout<<"\n\nNUMBER : "<< head -> number;
```

```

cout<<"\nADDRESS : "<< head -> address;
cout<<"\nE-MAIL : "<< head -> email;
inorderinside(head -> mid);
inorderinside(head -> right);
}

}

void load(){
struct node *new__node;
int hash ;
char name[30];
char number[15];
char address[40];
char email[40];
char c[40];
cout<<"\n\n\n\tPlease wait for a moment\n";
cout<<"\tRetrieving data form file";
char fn[] = "Directory_A.txt";

for(int pp=0;pp<4;pp++)
{
Sleep(500);
cout<<".";
}
cout<<"\n\n";

```

```

for(char i = 'A'; i <= 'Z' ; ++i){
fn[10] = i;
hash = int(i) % 65;

fstream ob(fn);
while(!ob.eof()){
ob.getline(c, 40);
if(c[0] != '\0' ){
strcpy(name, c);
ob.getline(c, 40);
strcpy(number, c);
ob.getline(c, 40);
strcpy(address, c);
ob.getline(c, 40);
strcpy(email, c);

new__node = (struct node*)malloc(sizeof(struct node));

strcpy(new__node -> name, name);
strcpy(new__node -> number, number);
strcpy(new__node -> address, address);
strcpy(new__node -> email, email);
head[hash] = insertinside(head[hash],new__node);
}
}
ob.close();
ob.open(fn,ios::trunc);

```

```

ob.close();
}
}

struct node *input(struct node *ne_node){

char name[30];
char number[15];
char address[40];
char email[40];
int k = 0;

cin.getline(name,30);

while((strlen(name)==0)){

cout<<"\nENTER VALID NAME : ";
cin.getline(name, 30);

}

while(name[k] != '\0'){
name[k] = toupper(name[k]);
k++;
}

```

```
cout<<"\nENTER NUMBER : ";
```

```
cin.getline(number,15);
```

```
while(strlen(number)!=10){
```

```
    cout<<"\nENTER NUMBER : ";
```

```
cin.getline(number,15);
```

```
}
```

```
cout<<"\nENTER ADDRESS : ";
```

```
cin.getline(address,40);
```

```
cout<<"\nENTER E-MAIL : ";
```

```
cin.getline(email,40);
```

```
ne_node = (struct node*)malloc(sizeof(struct node));
```

```
strcpy(ne_node -> name, name);
```

```
strcpy(ne_node -> number, number);
```

```
strcpy(ne_node -> address, address);
```

```
strcpy(ne_node -> email, email);
```

```
return ne_node;
```

```
}
```

```
struct node* deleteNodeinside(struct node* root, char key[],char  
number[])
```

```

{
struct node* temp;
char mob_num[15];
if (root== NULL){
    return root;
}

    if (strcmp(root -> name,key) > 0){
        root->left = deleteNodeinside(root->left, key,number);
    }

    else if (strcmp(root -> name,key) < 0){
        root->right = deleteNodeinside(root->right, key,number);
    }

    else {

        if(strcmp(root -> number,number) == 0){

            if(root->mid != NULL){
                temp = findmin(root->mid);
                strcpy(root -> name,temp -> mid-> name);
strcpy(root -> number,temp -> mid-> number);
strcpy(root -> address,temp -> mid-> address);
strcpy(root -> email,temp -> mid-> email);
deletestackpush(temp->mid);
                free(temp->mid);
temp -> mid = NULL;

```

```

    return root;

    }

}

if(strcmp(root -> number,number) != 0){
temp = root;
while(strcmp(temp-> mid ->number,number)!=0){
temp=temp->mid;
}deletestackpush(temp->mid);
free(temp-> mid);
if(temp -> mid -> mid !=NULL){
temp -> mid = temp -> mid -> mid;
}
else{temp -> mid =NULL;}

return temp;
}

    if (root->left == NULL) {
        temp = root->right;
        deletestackpush(root);
        free(root);
        return temp;
    }

    else if (root->right == NULL) {
        temp = root->left;
        deletestackpush(root);
        free(root);
        return temp;
    }

```



```
}
```

```
temp = findmin(root->right);
```

```
strcpy(root -> name,temp -> name);
```

```
strcpy(root -> number,temp -> number);
```

```
strcpy(root -> address,temp -> address);
```

```
strcpy(root -> email,temp -> email);
```

```
root->right = deleteNodeinside(root->right, temp->name,temp ->  
number);
```

```
deletestackpush(root);
```

```
}
```

```
return root;
```

```
}
```

```
struct node* findmin(struct node* tree)
```

```
{
```

```
struct node* node1 = tree;
```

```
if(tree -> mid ==NULL){
```

```
while (node1 != NULL && node1->left != NULL)
```

```
node1 = node1->left;
```

```
return node1;
```

```
}
```

```

else if(tree -> mid !=NULL){
while (node1 != NULL && node1->mid ->mid!= NULL)
    node1 = node1->mid;
    return node1;
}

}

```

```

void insertstackpush(struct node* temp){
struct node* newnode;
newnode = (struct node*) malloc(sizeof(struct node));

strcpy(newnode -> name,temp -> name);

strcpy(newnode -> number,temp -> number);
strcpy(newnode -> address,temp -> address);
strcpy(newnode -> email,temp -> email);
newnode->left = insertstack;
insertstack = newnode;

}

```

```

void display() {
    struct node* ptr;
    if(insertstack==NULL)
        cout<<"stack is empty";
    else {
        ptr = insertstack;

```

```

    cout<<"Stack elements are: ";
    while (ptr != NULL) {
        cout<< ptr->name <<" ";
        ptr = ptr->left;
    }
}
}

void deletestackpush(struct node* temp){
    struct node* newnode;
    newnode = (struct node*) malloc(sizeof(struct node));

    strcpy(newnode -> name,temp -> name);
    strcpy(newnode -> number,temp -> number);
    strcpy(newnode -> address,temp -> address);

    strcpy(newnode -> email,temp -> email);

    newnode->left = deletestack;
    deletestack = newnode;

}

void gosave(){
    fstream ob;
    char fn[] = "Directory_A.txt", c[40];
    int i = 0,hash,j;

```

```

cout<<"\nSAVING TO FILE";
for(int pp=0;pp<4;pp++)
{
cout<<".";
Sleep(500);
}
cout<<"\n";
while(insertstack !=NULL){

fn[10] = insertstack -> name[0];
ob.open(fn,ios::app);
ob<<"\n"<<insertstack -> name<<"\n"<<insertstack ->
number<<"\n"<<insertstack -> address<<"\n"<<insertstack -> email;
ob.close();
insertstack = pop(insertstack);

}

while(deletestack != NULL){

hash = int(deletestack -> name[0]) % 65;
fn[10] = deletestack -> name[0];
ob.open(fn,ios::out);
savein(head[hash],fn);
ob.close();
deletestack = pop(deletestack);
}
}

```

```

struct node* pop(struct node*stack) {
struct node* temp;
if(stack != NULL){

    temp = stack;

    stack = stack->left;
free(temp);
return stack;
}
}

```

```

void savein(struct node *head,char fn[]){
ofstream ob;

if(head != NULL){

ob.open(fn,ios::app);

ob<<"\n"<<head -> name<<"\n"<<head -> number<<"\n"<<head ->
address<<"\n"<<head -> email;

ob.close();

savein(head -> left,fn);
savein(head -> right,fn);
}
}

```

```
struct node* searchin(struct node* root,char name[],char number[])
```

```
{
```

```
if(root==NULL)
```

```
{
```

```
cout<<"\nNO SUCH SUBSCRIPTION FOUND\n";
```

```
return root;
```

```
}
```

```
else if(strcmp(root->name,name)==0)
```

```
{
```

```
if(strcmp(root->number,number)==0)
```

```
return root;
```

```
else
```

```
return searchin(root->mid,name,number);
```

```
}
```

```
else if(strcmp(root->name,name)<0)
```

```
{
```

```
return searchin(root->right,name,number);
```

```
}
```

```
else if(strcmp(root->name,name)>0)
```

```
{
```

```
return searchin(root->left,name,number);
```

```
}
```

```
}
```

```
void modify(struct node* ptr)
```

```
{
```

```
char a;
```

```
int hash1,hash,k=0;
```

```
struct node*ptr1 = (struct node*)malloc(sizeof(struct node));
```

```
do{
```

```
cout<<"\nWhich entity has to be modified??\nN - NAME\nA -  
ADDRESS\nP - PHONE NUMBER\nE - MAIL_ID\n";
```

```
cin>>a;
```

```
switch(a)
```

```
{
```

```
case ('N'):
```

```
char name[30];
```

```
cin.getline(name,30);
```

```
while(strlen(name)!=0)
```

```
{
```

```
cout<<"\nEnter the new name of subscriber: ";
```

```
cin.getline(name,30);
```

```

}

while(name[k] != '\0'){
    name[k] = toupper(name[k]);
    k++;
}

hash1 = int(name[0]) % 65;
hash = int(ptr -> name[0]) % 65;
strcpy(ptr1->name,name);
strcpy(ptr1->address,ptr->address);
strcpy(ptr1->number,ptr->number);
strcpy(ptr1->email,ptr->email);


head[hash1] = insertinside(head[hash1], ptr1);
print();
head[hash] = deleteNodeinside(head[hash],ptr -> name,ptr -> number);
print();


cout<<"\n\nNAME:::::"<<ptr1 -> name<<endl;
deletestackpush(ptr1);
deletestackpush(ptr);
break;


case ('A'):
    char address[40];

    cin.getline(address,40);

    while(strlen(address)==0)

```



```

{
    cout<<"\nEnter new address of subscriber: ";
    cin.getline(address,40);
}

strcpy(ptr->address,address);
deletestackpush(ptr);
break;

case ('P'):
    char no[20];

    cin.getline(no,20);

    while(strlen(no)!=10)
    {
        cout<<"\nEnter new phone number of subscriber: ";
        cin.getline(no,20);
    }

    strcpy(ptr->number,no);
    deletestackpush(ptr);
    break;

case ('E'):
    char mail[40];

    cin.getline(mail,40);

```

```

while(strlen(mail)==0)
{
    cout<<"\nEnter new E-mail ID of subscriber: ";
    cin.getline(mail,40);
}

strcpy(ptr->email, mail);
deletestackpush(ptr);
break;

default:
cout<<"\nUnrecognized character entered!\nKindly re-enter character
from list or press 'm\\M' to cancel MODIFY process\n";
cin>>a;

}

if(a!='m' && a!='M'){

cout<<"Modification still working... press 'm//M' to terminate, and to
process press 'y/Y'\n";
cin>>a;

}

}while(a!='M' && a!='m');

}

{
    cout<<"\nEnter E-mail ID of subscriber: ";
    cin.getline(mail,40);

```

```
}
```

```
strcpy(ptr->email, mail);
```

```
deletestackpush(ptr);
```

```
break;
```

```
default:
```

```
cout<<"\nUnrecognized character entered!\nKindly re-enter character  
from list";
```

```
}
```

```
}
```

9 OUTPUT

```
C:\Users\Karthick\Desktop\SK\tda.exe
*****PHONEBOOK*****

Please wait for a moment
Retrieving data from file....

MENU
1)INSERT
2)PRINT
3)SEARCH AND MODIFY
4)DELETE A CONTACT
5)EXIT
ENTER YOUR CHOICE
1

Please wait for a moment
Retrieving data from file....

MENU
1)INSERT
2)PRINT
3)SEARCH AND MODIFY
4)DELETE A CONTACT
5)EXIT
ENTER YOUR CHOICE
2

NAME : ALEX
NUMBER : 2345678904
ADDRESS : RTY
E-MAIL : SRS

NAME : BEN
NUMBER : 0000111110
ADDRESS : UQUP
E-MAIL : doqbc

NAME : DAVID
NUMBER : 9999999998
ADDRESS : P56 US
E-MAIL : DAVT@PP.COM

NAME : KARTHIK
NUMBER : 9999999999
ADDRESS : P56
E-MAIL : K@kmail.com

NAME : PITY
NUMBER : 0000000000
ADDRESS : PUNJAB
E-MAIL : TU

MENU
1)INSERT
2)PRINT
3)SEARCH AND MODIFY
4)DELETE A CONTACT
5)EXIT
ENTER YOUR CHOICE
3

ADDRESS : P56
E-MAIL : K@kmail.com

NAME : PITY
NUMBER : 0000000000
ADDRESS : PUNJAB
E-MAIL : TU

MENU
1)INSERT
2)PRINT
3)SEARCH AND MODIFY
4)DELETE A CONTACT
5)EXIT
ENTER YOUR CHOICE
4

ENTER VALID NAME : bin

ENTER NUMBER : 7975875458

ENTER ADDRESS : No:8766/98767,XXX,YYY

ENTER E-MAIL : hhdusthin@kmail.com

CONTINUE ? (y/n) n

MENU
1)INSERT
2)PRINT
3)SEARCH AND MODIFY
4)DELETE A CONTACT
5)EXIT
ENTER YOUR CHOICE
5

NAME : ALEX
NUMBER : 2345678904
ADDRESS : RTY
E-MAIL : SRS

NAME : BEN
NUMBER : 0000111110
ADDRESS : UQUP
E-MAIL : doqbc

NAME : BTH
NUMBER : 7975875458
ADDRESS : No:8766/98767,XXX,YYY
```

\\C:\Users\Karthick\Desktop\SD\sdta.exe

```
2)PRINT
3)SEARCH AND MODIFY
4)DELETE A CONTACT
5)EXIT
ENTER YOU CHOICE
2.

NAME : ADAM
NUMBER : 2345678904
ADDRESS : RTV
E-MAIL : sbg

NAME : BEN
NUMBER : 0000111110
ADDRESS : UGUP
E-MAIL : dsqpc

NAME : BIN
NUMBER : 7975875458
ADDRESS : No;6766/98767,XXX,YYY
E-MAIL : bldustbldngmkaill.com

NAME : DAVID
NUMBER : 9999876548
ADDRESS : PSG US
E-MAIL : DAVID@MP.COM

NAME : KARTHIK
NUMBER : 9999951436
ADDRESS : PURE
E-MAIL : K@psmail.com

NAME : PITY
NUMBER : 0000099998
ADDRESS : PUNJAB
E-MAIL : TU

MENU
1)INSERT
2)PRINT
3)SEARCH AND MODIFY
4)DELETE A CONTACT
5)EXIT
ENTER YOU CHOICE
```

```
1.
1)INSERT
2)PRINT
3)SEARCH AND MODIFY
4)DELETE A CONTACT
5)EXIT
ENTER YOU CHOICE
1.
ENTER THE SUBSCRIBER'S NAME TO BE SEARCHED
BIN
ENTER THE SUBSCRIBER'S NUMBER
0000111110

NAME : BEN
NUMBER : 0000111110
ADDRESS : UGUP
E-MAIL : dsqpc

Do you want to modify? (y/n)
Y

Which entity has to be modified??
N - NAME
A - ADDRESS
P - PHONE NUMBER
E - MAIL_ID
N

Enter the new name of subscriber: PORTMAN
```

\\C:\Users\Karthick\Desktop\SD\sdta.exe

```
N
Enter the new name of subscriber: PORTMAN

DIRECTORY AFTER MODIFICATION

NAME : ADAM
NUMBER : 2345678904
ADDRESS : RTV
E-MAIL : sbg

NAME : DAVID
NUMBER : 9999876548
ADDRESS : PSG US
E-MAIL : DAVID@MP.COM

NAME : KARTHIK
NUMBER : 9999951436
ADDRESS : PURE
E-MAIL : K@psmail.com

NAME : PITY
NUMBER : 0000099998
ADDRESS : PUNJAB
E-MAIL : TU

NAME : PORTMAN
NUMBER : 0000111110
ADDRESS : UGUP
E-MAIL : dsqpc

NAME:::PORTMAN
Modification still working... press 'n/N' to terminate, and to process press 'y/Y'
```

```
MENU
1)INSERT
2)PRINT
3)SEARCH AND MODIFY
4)DELETE A CONTACT
5)EXIT
ENTER YOU CHOICE
4.
ENTER THE NAME
PITY
ENTER THE NUMBER
0000099998

MENU
1)INSERT
2)PRINT
3)SEARCH AND MODIFY
4)DELETE A CONTACT
5)EXIT
ENTER YOU CHOICE
3.

NAME : ADAM
NUMBER : 2345678904
ADDRESS : RTV
E-MAIL : sbg

NAME : DAVID
NUMBER : 9999876548
ADDRESS : PSG US
E-MAIL : DAVID@MP.COM
```

```

MENU
1)INSERT
2)PRINT
3)SEARCH AND MODIFY
4)DELETE A CONTACT
5)EXIT
ENTER YOU CHOICE
2.

NAME : ADAM
NUMBER : 2345678904
ADDRESS : RTV
E-MAIL : SBC

NAME : DAVID
NUMBER : 9999876548
ADDRESS : PSG US
E-MAIL : DAVID@PP.COM

NAME : KARTHIK
NUMBER : 9999951436
ADDRESS : PUNE
E-MAIL : K@emall.com

NAME : PORTMAN
NUMBER : 0000111110
ADDRESS : UKUP
E-MAIL : doplc

MENU
1)INSERT
2)PRINT
3)SEARCH AND MODIFY
4)DELETE A CONTACT
5)EXIT
ENTER YOU CHOICE

```

```

ENTER YOU CHOICE
MENU
1)INSERT
2)PRINT
3)SEARCH AND MODIFY
4)DELETE A CONTACT
5)EXIT
ENTER YOU CHOICE
5.

SAVING TO FILE....

.....
Process exited after 162.9 seconds with return value 0
Press any key to continue . . .

```

```

MENU
1)INSERT
2)PRINT
3)SEARCH AND MODIFY
4)DELETE A CONTACT
5)EXIT
ENTER YOU CHOICE
1.

ENTER VALID NAME : KARTHIK

ENTER NUMBER : 5766867893

ENTER ADDRESS : RRR

ENTER E-MAIL : TR@OUTLOOK.COM

CONTINUE ? (y/n)      N

MENU
1)INSERT
2)PRINT
3)SEARCH AND MODIFY
4)DELETE A CONTACT
5)EXIT
ENTER YOU CHOICE
2.

NAME : ADAM
NUMBER : 2345678904
ADDRESS : RTV
E-MAIL : SBC

NAME : DAVID
NUMBER : 9999876548
ADDRESS : PSG US
E-MAIL : DAVID@PP.COM

NAME : KARTHIK
NUMBER : 9999951436
ADDRESS : CHENNAI
E-MAIL : K@emall.com

NAME : KARTHIK
NUMBER : 5766867893
ADDRESS : RRR
E-MAIL : TR@OUTLOOK.COM

MENU
1)INSERT
2)PRINT
3)SEARCH AND MODIFY
4)DELETE A CONTACT
5)EXIT
ENTER YOU CHOICE
1.

ENTER VALID NAME : PORTMAN

ENTER NUMBER : 0000111110

ENTER ADDRESS : UKUP

ENTER E-MAIL : doplc

Already exist!!!!

CONTINUE ? (y/n)      N

```

INTERPRETATION OF RESULTS:

- The output mentioned – above indicates the working process of the ternary search tree with hash table for various processes like insertion, deletion, search, sort and modify.
- The outputs are checked for test cases which includes the duplicate entries with same name and number
- The phone book directory system can handle all types of user data efficiently and is compatible with file data handling.
- Using both files(.txt) and dynamic memory, the data is stored and accessed.
- Also, the added advantage of the system is use of a stack to record the modifications in the data during program run and thus the data need not be copied, erased and stored every time in the text files.

10.LIMITATIONS:

Even though, identifying the duplicate values is a key problem that has been solved. The space wastage occurs for each creation of mid node will increase linearly.

Using tree data structure and also stack data structure used to store data temporarily will take up more space in memory albeit, a rise in time efficiency.

In our project, we have focused mainly on optimizing our search and modify algorithm, compared to deletion of a node, the deletion operation may take up more time than a standard time complexity depending upon the situation.

11. APPLICATIONS AND ADVANTAGES:

Ternary search trees can be used to solve many problems in which a large number of strings must be stored and retrieved in an arbitrary order. Some of the most common or most useful of these are below:

- Can be used as a quick and space-saving data structure for mapping strings to other data.
- Can also be integrated to implement auto-completion operation.
- We can even perform spell check operation using TST
- Near-neighbour searching program (of which spell-checking is a special case)
- Can be used as a database especially when indexing by several non-key fields is desirable.
- Can also be used in place of a hash table.

12. CONCLUSION

Thus, using the TST algorithm, the phone book directory was implemented in C++ programming language. The obtained results were satisfactorily close to the expected outcomes and the main motive to reduce the time complexity of sorting in phone book directory with large amount of data was preserved.

13. REFERENCES:

We also used GitHub repository for sharing and copying of code. Visit the repository following the link below:

- <https://github.com/Hari545543/Phonebook-Management-system-using-data-structures/blob/main/PhoneBook%20using%20BST/c%2B%2Bphonecopy.cpp>
- <https://www.geeksforgeeks.org/binary-search-tree-data-structure>
- <https://www.geeksforgeeks.org/ternary-search-tree>
- https://en.wikipedia.org/wiki/Ternary_search_tree

14.BIODATA:

NAME: HARI KRISHNA S

REG. NO: 20BEC1004

E-MAIL ID: hari.krishna2020@vitstudent.ac.in



Hari Krishna S

20BEC1004



NAME: HARISH D

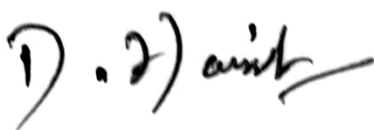
REG. NO: 20BEC1034

E-MAIL ID: harish.d2020@vitstudent.ac.in



Harish D

20BEC1034



NAME: GABRIEL K
REG. NO: 20BEC1320
E-MAIL ID: gabriel.2020@vitstudent.ac.in



Gabriel K
20BEC1320

Gabriel

NAME: KARTHIKEYAN S
REG. NO: 20BEC1241
E-MAIL ID: karthikeyan.s2020a@vitstudent.ac.in



Karthikeyan S
20BEC1241

S Karthikeyan