ENGLISH LEXICON

PROJECT REPORT

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

Of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

By

Aditi Agrawal-03401012019
Balpreet Kaur-05201012019
Jaskiran Kaur-05301012019
Heena Ahmed-05401012019
Saumya Jangid-07301012019

Guided by

Dr.Vivekanand Jha
Assistant Professor
Indira Gandhi Delhi Technical University for Women

INDIRA GANDHI DELHI TECHNICAL UNIVERSITY FOR WOMEN
NEW DELHI – 110006

UNDERTAKING REGARDING ANTI-PLAGIARISM

We hereby, declare that the material/ content presented in the report are free from plagiarism and is properly cited and written in our own words. In case, plagiarism is detected at any stage, we shall be solely responsible for it. A copy of the Plagiarism Report is also enclosed at the end.

Aditi Agrawal

(03401012019)

Balpreet Kaur

(05201012019)

Jaskiran Kaur

(05301012019)

Heena Ahmed

(05401012019)

Saumya Jangid

(07301012019)

ACKNOWLEDGEMENT

We would like to acknowledge the contributions of the following people; without whose help and guidance this report would not have been completed.

I express my sincere thanks to **MRS. AMITA DEV**, Honorable Vice Chancellor, Indira Gandhi Delhi Technical University for Women, Kashmere Gate.

I also acknowledge the counsel and support of our teacher and mentor**, DR.VIVEKANAND JHA,** Assistant Professor, Indira Gandhi Delhi Technical University for Women, Kashmere Gate, with respect and gratitude, whose expertise, guidance, support, encouragement, and enthusiasm has made this report possible. Their feedback vastly improved the quality of this report and provided an enthralling experience. We are indeed proud and fortunate to be supported by him. His constant encouragement, valuable suggestions and moral support and blessings.

Although it is not possible to name individually, we shall ever remain indebted to the faculty members of our university as well for their persistent support and cooperation extended during this work.

Aditi Agrawal

(03401012019)

Balpreet Kaur

(05201012019)

Jaskiran Kaur

(05301012019)

Heena Ahmed

(05401012019)

Saumya Jangid

(07301012019)

DECLARATION

We solemnly declare that the project report on **ENGLISH LEXICON,** is based on our own work carried out during the course of our study under the supervision of **DR. VIVEKANANDA JHA,** Assistant Professor, Indira Gandhi Delhi Technical University for Women. We assert the statements made and conclusions drawn are an outcome of our research work. We further certify that:

I. The work contained in the report is original and has been done by me under the supervision of my supervisor.
II. The work has not been submitted to any other Institution for any other degree/diploma/certificate to any other University of India or abroad.
III. We have followed the guidelines provided by the university in writing it.
IV. Whenever we have used materials (text, data, theoretical analysis/equations, codes/program, figures, tables, pictures, text etc.) from other sources, we have given due credit to them in the report.

Aditi Agrawal

(03401012019)

Balpreet Kaur

(05201012019)

Jaskiran Kaur

(05301012019)

Heena Ahmed

(05401012019)

Saumya Jangid

(07301012019)

ABSTRACT/SUMMARY

The principal fundamental data structures and algorithms used in the field of computer science such as arrays trees (Binary search trees), Linked lists, etc are covered into a coherent framework. Important focus has been given to BST to implement the project. It brings together a broad range of topics of trees in C++ language. The lexicon is implemented using BST with efficient research work. Data structures(Trees) are formulated to represent various types of information/data in such a way that it can be conveniently and efficiently manipulated by the algorithms that are required to be developed in the industry. Throughout this report, the recurring practical issues of algorithm specification, verification and performance has been analysed.

INDEX

CHAPTER 1: INTRODUCTION

Computer is a powerful electronic machine which is basically used for data processing and manipulation. We can use it to store data and also manipulate it to create real life applications, using languages or softwares. Dictionary is a basic human need, in which a person needs to look up words and analyse their meanings, antonyms, synonyms, etc. which are required in the industry. So, our objective is to create a English Lexicon, which contains so many words, a user can search for.

1.1 Implementation of Lexicon

The implementation of a Lexicon requires writing a required set of procedures (algorithms) that create and also manipulate instances of that specific structure. The observation motivates the theoretical concept of an abstract data type, a data structure, binary search trees, and the also mathematical properties of the operations of trees. (including their space and time cost). Implementation of English Lexicon can be done using C++ language. The lexicon can read a word, display the word, insert and create a word, delete the word, etc.

1.2 C++ Language

C++ was developed by Bjarne Stroustrup. It is an extension to the C language. It's a portable high level language that can be used to develop applications which can be adapted in multiple platforms. It's one of the most popular languages across the world. The C++ programming language was initially standardized in 1998 as ISO/IEC 14882:1998, but it was then changed to C++03, C++11, C++14, and C++17 standard.

CHAPTER 2: BASICS OF TREES

Data Structures are broadly divided into two categories:

1) Primitive Data Structure
2) Non-Primitive Data Structure

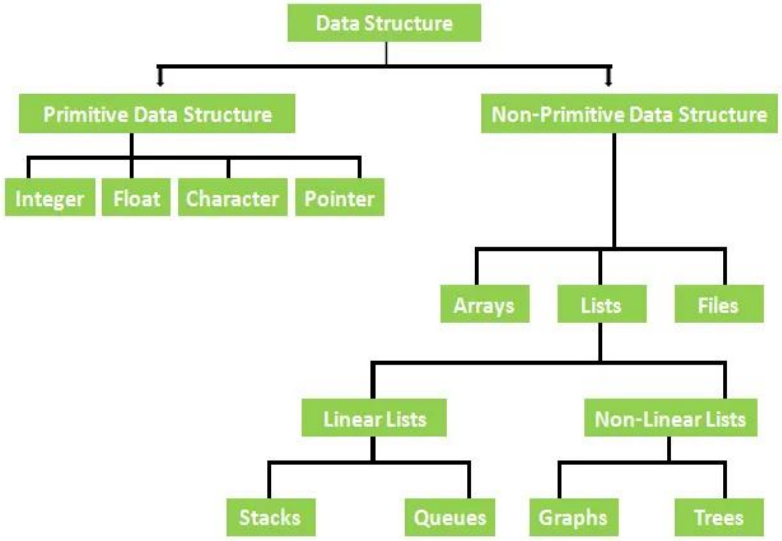


Figure 1: Classification of Data Structures

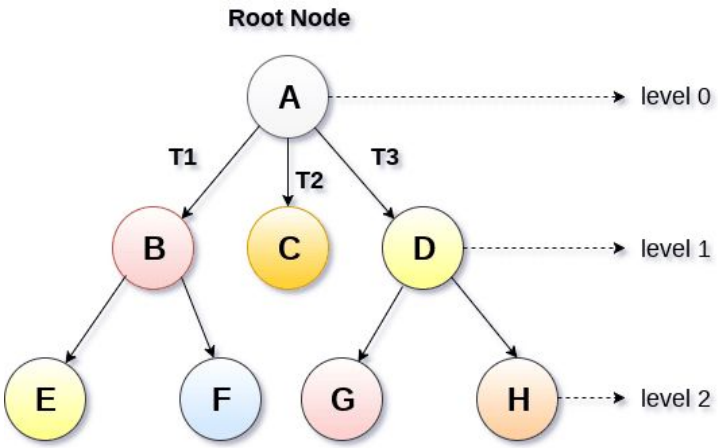
2.1 Definition

- A Tree is a data structure which works on recursion having the set of one or multiple nodes having data where one node is known as the root of the tree whereas the remaining nodes are called as the children of the root.
- The nodes other than the root node are divided in the sets which are non-empty and each one of them is defined as sub-tree.
- A parent-child relationship is maintained between the nodes.
- A tree can be traversed in many ways like in-order, post- order, pre-order, etc.

2.2 Basic terminology

- **Root Node**:- The topmost node in the tree hierarchy is called the root. In other words, It doesn't have any parent.

- **Sub Tree** :- If the root node is not null, the tree T1, T2 and T3 is called sub-trees of the root node.
- **Leaf Node** :- Leaf node is the node which doesn't have any child node.. Leaf node is the also called the bottom most node of the tree.. Leaf nodes can also be called external nodes.
- **Path** :- The sequence of consecutive edges is called path.
- **Ancestor node** :- An ancestor of a node is any predecessor node on a path from root to that node. The root node doesn't have any ancestors.
- **Degree** :- The number of children, a node have is termed as degree.. Degree of a leaf node(i.e this parent has 0 children or null values are stored in its children) is always 0 while in a complete binary tree, degree of each node is equal to 2.
- **Level Number**:- A level number is assigned in such a way that each node is present at one level higher than its parent. As seen in the figure below, Root node of the tree is always present at level 0.



Tree

2.3 Types of Trees

There are many types of trees, some of which are:

1.Generic Trees:

- In a generic tree, A node can have any number of children nodes but it can have only a single parent. This tree is the super-set of all other trees.

2. Binary Trees:

- In a binary tree, A node can have at most two children nodes for each parent., called left child and right child respectively.

3. Binary Search Trees:

Binary Search Tree (BST) is a binary tree in which The left child value of a node should be less than or equal to the parent value and the right child value should always be greater than or equal to the parent's value.

2.4 Applications of Binary Search Trees:

There are various applications of Binary Search trees because of being efficient in terms of searching and reducing the time for it to log n. One of the main applications of BST is to implement a dictionary in which words can be inserted. Dictionaries can be implemented using a binary search tree. A binary search tree is a binary tree such that each node stores a key of a dictionary. Word 'W' of a node is always greater than the words present in its left sub tree. Similarly, the word 'W'' of a node is always lesser than the words present in its right subtree. **Implementation of English lexicon** can be done efficiently which will read a list of words and their definitions. Users can look for their meaning and their synonyms. Insertion of new words, searching and deletion, and several other modularity and actions can also be performed.

CHAPTER 3: CREATION OF A NODE IN LEXICON

To add new words in English Lexicon , we first have to create new nodes in the binary search tree. This function is used to create a new node of binary search tree such that each node stores a key(i.e word along with its meaning, antonym and synonym) of a dictionary.
The memory to the new node is allocated dynamically using the malloc function. Any number of nodes can be created and linked to the existing node.

4.1 Algorithm for creation

The following steps depicts how the function is created and implemented.

- Structure definition and root creation

```
struct BSTnode {
   char word[128];
   char meaning[256];
   char antonym[256];
   char synonym[256];
   struct BSTnode *left, *right;
};

struct BSTnode *root = NULL;
```

- Creation of new node

```
struct BSTnode * createNode(char *word, char *meaning, char *synonym, char
*antonym)
{
```

1. struct BSTnode *newnode
2. Allocate the memory to the newnode dynamically using the malloc function
   newnode = (struct BSTnode *)malloc(sizeof(struct BSTnode))
3. Assign the values to the respective fields of newnode using the strcpy function
   strcpy(newnode->word, word)
4. Set Left and Right pointers of newnode = NULL
   newnode->left = newnode->right = NULL;
5. return newnode;

```
}
```

4.2 Pseudo Code for the Creation Module

```cpp
#include <iostream>
#include <stdlib.h>
#include <cstring>
using namespace std;

struct BSTnode {
    char word[128];
    char meaning[256];
    char antonym[256];
    char synonym[256];
    struct BSTnode *left, *right;
};

struct BSTnode *root = NULL;


struct BSTnode * createNode(char *word, char *meaning, char *synonym, char *antonym)
/*This function is used to create a new node of binary search tree such that each node stores a key(i.e word along with its meaning, antonym and syn
The memory to the new node is allocated dynamically using the malloc function. Any number of nodes can be created and linked to the existing node.*/

{
    struct BSTnode *newnode;
    newnode = (struct BSTnode *)malloc(sizeof(struct BSTnode));
    strcpy(newnode->word, word);
    strcpy(newnode->meaning, meaning);
    strcpy(newnode->synonym, synonym);
    strcpy(newnode->antonym, antonym);
    newnode->left = newnode->right = NULL;
    return newnode;
}
```

4.3 Time and space Complexity

The time complexity for new node creation is O(1) in both average and worst case.

The space complexity for new node creation is O(1) in both average and worst case.

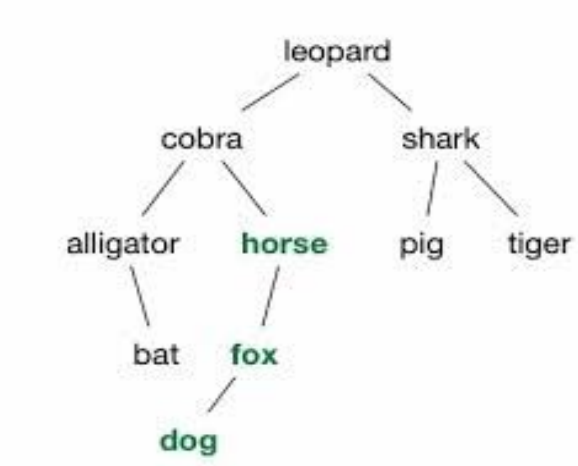
CHAPTER 4: INSERTION IN LEXICON

To insert in the English lexicon, we call the module ‘create’, which basically allocates the memory to the ‘word’ node dynamically. The word node consists of the inputted word, its meaning, its synonym, and its antonym. The strcasecmp() function, a string library function, is used to compare, while ignoring differences in case, the string pointed to by word 1(string1) to the string pointed to by word 2(string 2). All alphabetic characters in *string1* and *string2* are converted to lowercase before comparison i.e. without sensitivity. The strcasecmp() function operates on null terminated strings. The string arguments to the function are expected to contain a null character ('\0') marking the end of the string.

The function returns a value indicating the relationship between the two strings, as follows:

Value	**Meaning**
Less than 0	*string1* less than *string2*
0	*string1* equivalent to *string2*
Greater than 0	*string1* greater than *string2*

Table 1. Return values of strcasecmp()



4.1 Algorithm for insertion

1. To insert a word, we first check if the root is null or not i.e if the tree is empty, we create the root node by calling the create function.
2. If the root is not null, we iterate through the tree while we don't reach the leaf node. First step is to compare the word we want to insert with the current word present in the tree, if it's equal, a message is displayed that it's a duplicate entry.
3. Otherwise we make the current node as a parent node and check the left and right subtree.
4. If the result is greater than zero, i.e the word to be inserted is greater than the current word, we shift to right, else we shift to the left subtree (when the value is less than 0).
5. Now, when current points to a null pointer, we allocate memory for the new node, and position the new word according to the properties of BST.
6. We insert the word in the parent's right if the result is greater than zero, and make it the right child and if not, we insert in the parent's left, making it the left child and that's how we insert the word in our dictionary, the English Lexicon.

4.2 Pseudo Code for the Module

```
void insert(char *word, char *meaning, char *synonym, char *antonym)
{
    struct BSTnode *parent = NULL, *current = NULL, *newnode = NULL;
    int result = 0;
    if (root==NULL) {
        root = createNode(word, meaning , synonym , antonym);
        return;
    }
    current=root;
    while(current!=NULL){
        result = strcasecmp(word, current->word);
        if (result == 0) {
          //-------------------------------------------------------------------
    //The Dictionary contains a word, its meaning, its synonyms, antonyms….It cannot contain any duplicate words"
        cout<<"You have entered the same word again! It's a  duplicate entry. Please enter the new word for the dictionary"<<endl;
            return;
        }
        parent = current;
        if(result > 0){
            current= current->right;
        }

        else{
        current=current->left;
        }

    } //while loop ends here

    newnode = createNode(word, meaning, antonym, synonym);
    if(result>0){
        parent->right=newnode;
    }else{
        parent->left = newnode;
    }

    return;
} //function ends here
```

4.3 Time and Space Complexity

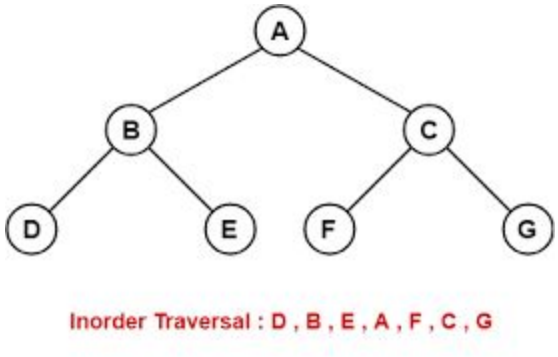
1. The insert function requires time proportional to the height of the tree(n) i.e log n(base 2) or n
2. . It takes time to execute O(log n) in the average case and O(n) time in the worst case (in case of skewed binary search tree).
3. The Space Complexity is O(n), where n is equal to the no. of words, i.e. the number of nodes in the lexicon.

CHAPTER 5: Traversal of Lexicon

We call this function to traverse and display all the words of our Lexicon along with its meaning, antonym and Synonym.

To print all the words of our lexicon, we make use of In-order traversal of binary search tree.The InOrder traversal is also known as left-node-right or left-root-right traversal or LNR traversal algorithm

We traverse our left subtree which consists of all the words that are smaller than our root word(i.e key) which we have distinguished using inbuilt strcasecmp function. Then we print our root node and finally traverse the left sub tree which has all the words that are greater than our root node. This helps us to print all the elements of our dictionary in alphabetical order.



Inorder traversal illustration

5.1 Algorithm of insertion

1. We repeat our steps 2 through until the root node is not NULL.
2. Then we call our function recursively to traverse the left subtree which has words smaller than the root node
3. Then we print the word, its meaning, synonyms and antonym of our root node

4. And then finally we traverse the right subtree which has all the words which are greater than the root root node

5.2 Pseudo Code for the Module

```
void inorderTraversal(struct BSTnode *myNode)
{
    if (myNode!=NULL)
    {
        inorderTraversal(myNode->left);

        cout<<"Word is "<<myNode->word<<endl;
        cout<<"Meaning: "<<myNode->meaning<<endl;
        cout<<"Synonym: "<<myNode->synonym<<endl;
        cout<<"Antonym: "<<myNode->antonym<<endl;

        inorderTraversal(myNode->right);
    }
    return;
}
```

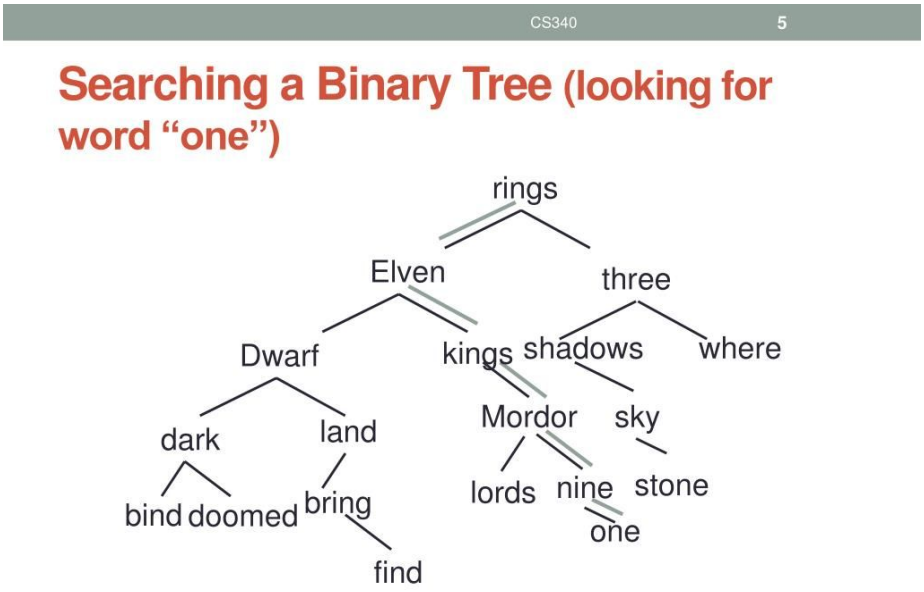
5.3 Time and Space Complexity

1. For the worst case time complexity, if a tree has n nodes, then each of the node is traversed once or a constant amount of work is done for each node. Therefore the time complexity is O(n). Average time complexity is O(n) and best case time complexity to traverse the tree is constant, that is O(1).

2. The Space required is O(1), except the recursion stack.

CHAPTER 6: Searching a word in Lexicon

Lexicon , an English Dictionary is helpful if we are able to search the required word faster . The data structure, Binary Tree helps in searching a given node (here, the required word) with Time complexity of O(h) for the worst case scenario , where h is the height of the tree . To search a Word in a Lexicon , we can call a module named search() and pass string (Word) to be searched in the lexicon in the module as a parameter . A new temp node is created which points to the root node of the lexicon . This temp node iterates over each node and checks if the word in the node is equal to the word to be searched . The nodes in the binary tree have word , synonym , antonym of the word . For searching we compare the given word with the nodes word while ignoring the case .ie. Lower case or upper case doesn't affect the searching . To serve the purpose we use in build strcasecmp() function



6.1 Algorithm for searching

- Word to be searched is passed as a string parameter in module Search()
- temp node is created ,temp node is used to iterate in tree using while loop
- iteration is performed until either we reach the leaf nodes or the word is found
- flag variable is created to check if the word is found and if yes , break the loop , initially it is set to value 0
- res variable is created to store the value obtained from strcasecmp() function , .ie. less than 0 , 0 , more than 0 , if the value is less than 0 , than the word in node is less than word to be searched , if the value is more than 0 then the word in node is more than word to be searched , if the word is equal to 0 than the word in node is same as word to be searched .
- in the while loop we check for condition when res is equal to zero , using if condition
- when the condition is reached we print the word , meaning , synonym , antonym , break the loop and set flag to 1
- else if , the res is greater than zero we search in right tree
- else if , the res is less than zero we search in left tree
- if we have traversed the whole tree and res has never been equal to zero than we come out of loop
- if the flag is zero , .ie. the word to be found is not in tree(Dictionary) then we print , word not found and return

6.2 Pseudo Code for the Search module

```
struct node* search( char *str)
{

 /*This function searches the word in our lexicon , checking the left and right subtree, iteratively.
    It uses the string library function to compare the words according to their ASCII values,
    ignoring the case(upper or lowercase characters)
    */
    struct BSTnode *temp =NULL;
    int flag =0, res=0;
    if(root ==NULL){
        cout<<"Lexicon is empty";
    }
    temp = root;
    while(temp){
        if((res = strcasecmp(temp->word,str))==0){
            cout<<"word:"<<str<<endl;
            cout<<"Meaning:"<<temp->meaning<<endl;
            cout<<"Synonym:"<<temp->synonym<<endl;
            cout<<"Antonym:"<<temp->antonym<<endl;
            flag = 1;
            break;
        }
        temp = (res>0) ? temp ->left : temp ->right;
    }
    if(flag == 0){
        cout<<"word not found"<<endl;
    }
    return 0;
}
```

6.3 Time and Space Complexity

Time Complexity:

Best case : O(1) when the root node is same as root to be searched

Average case : O(logn) when any intermediate node is the node to be searched

Worst case : O(logn) when the leaf node is the node to be searched

Space Complexity:

In all cases only res , flag , and temp variable are made of constant size .ie. sizeof(int)

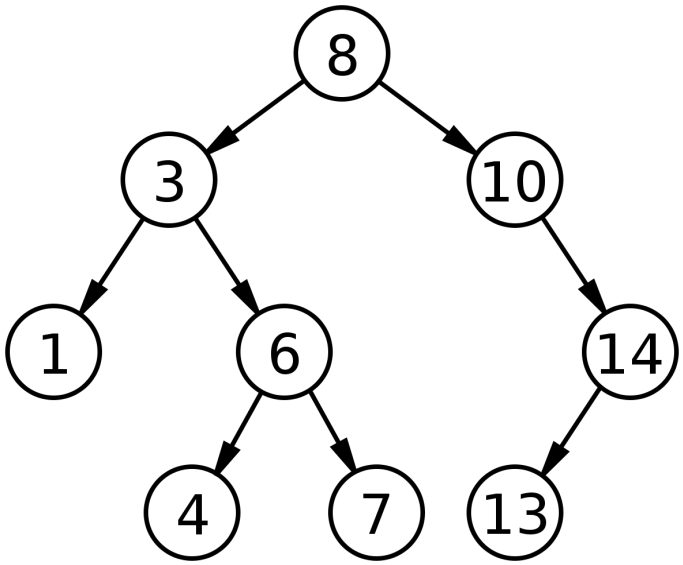
Best case : O(1)

Average case : O(1)

Worst case : O(1)

CHAPTER 7 : DELETION IN LEXICON

To delete a word in the English lexicon, we call the module ‘deletion’, which basically finds and deletes the requested word entered by the user . The word node consists of the inputted word, its meaning, its synonym, and its antonym. The strcasecmp() function, a string library function, is used to compare, while ignoring differences in case, the string pointed to by word 1(string1) to the string pointed to by word 2(string 2). All alphabetic characters in *string1* and *string2* are converted to lowercase before comparison i.e. without sensitivity. The strcasecmp() function operates on null terminated strings. The free() function is used to deallocate the memory allocated to the variable so that it can be used for other purposes.The argument of the function free ( ) is the pointer to the memory which is to be released. The ‘delete’ word can not be used as the function name as delete is a keyword that are predefined reserved words.



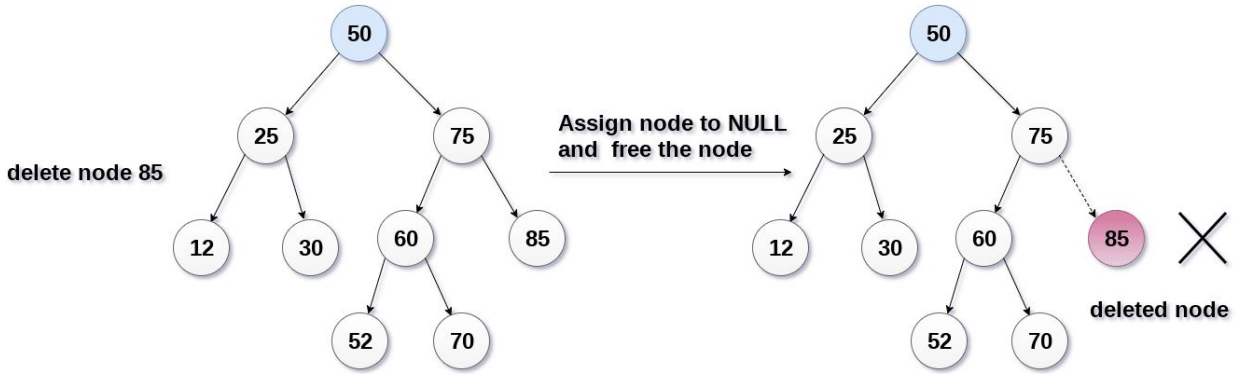
7.1 Algorithm of deletion

To delete a word from the dictionary (where words are stored in the form of a binary search tree), we first check the current word node, that is the root node of the tree. If the root node is null, we print on the console about its absence and the control of the program goes to the next executable statement. If the root node is not null, it implies we have at least one node or word present in the dictionary. We’ll iterate through the tree while we reach the leaf node and alongside we’ll search for the word to be deleted from the tree.If we’ll reach the leaf node

without finding the word, a message will be displayed stating that the word to be deleted is not present in the tree. If the word is found, then three cases are possible to happen with the word to be deleted.

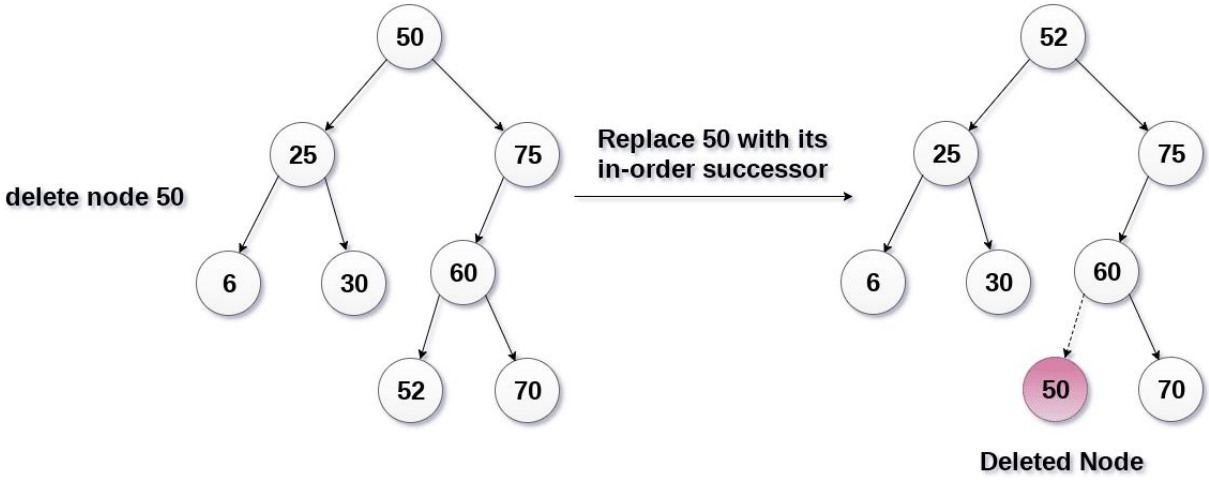
The three cases and the way they are handled is as follows :

- Case-1 : The word node is a leaf node



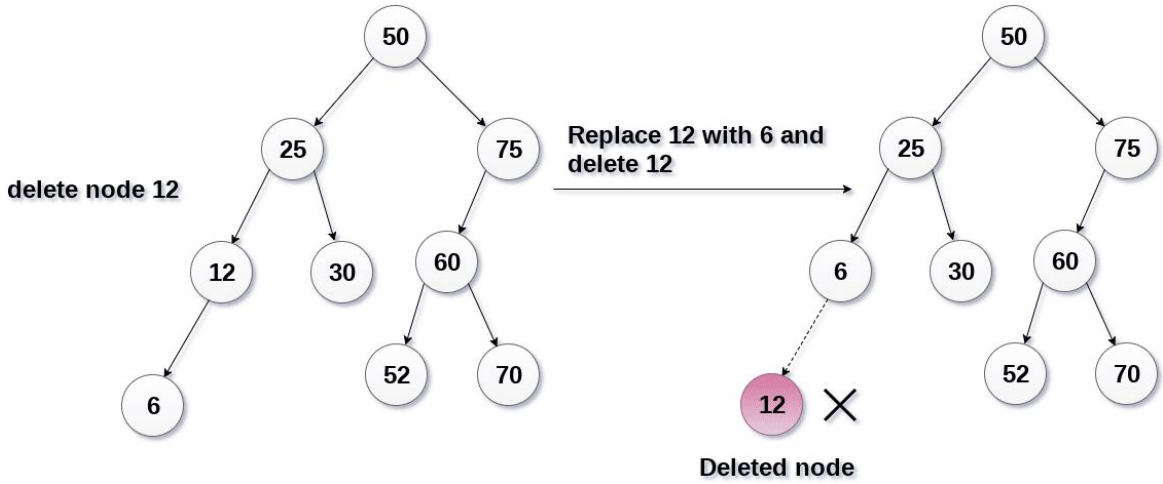
That word node is deleted by freeing the node.

- Case-2 : The word node is the node with two children nodes.



The inorder successor from the right subtree is found and the word of the node is replaced with it. The inorder successor is deleted by freeing the memory using a free function.

- Case-3 : The word node is the node with one children node.



A temporary node is created to save the child node and after deleting by freeing the node it is copied as the leaf node to the previous word node .

This is how we delete the requested word from the English Lexicon.

7.2 Pseudo Code for the Deletion Module

```
void deleteNode(char *str) {
        struct BSTnode *parent = NULL, *current = NULL, *temp = NULL;
        int flag = 0, res = 0;
        if (!root) {
                cout<<"Dictionary is empty!!"<<endl;
                return;
        }
        current = root;
        while (1) {
                res = strcasecmp(current->word, str);
                if (res == 0)
                        break;
                flag = res;
                parent = current;
                current = (res > 0) ? current->left : current->right;
                if (current == NULL)
                        return;
        }
        /* deleting leaf node */
        if (current->right == NULL) {
                if (current == root && current->left == NULL) {
                        free(current);
                        root = NULL;
                        return;
                } else if (current == root) {
                        root = current->left;
                        free (current);
                        return;
                }

                flag > 0 ? (parent->left = current->left) :
                                (parent->right = current->left);
```

```
        } else {
                /* delete node with single child */
                temp = current->right;
                if (!temp->left) {
                        temp->left = current->left;
                        if (current == root) {
                                root = temp;
                                free(current);
                                return;
                        }
                        flag > 0 ? (parent->left = temp) :
                                        (parent->right = temp);
                } else {
                        /* delete node with two children */
                        struct BSTnode *successor = NULL;
                        while (1) {
                                successor = temp->left;
                                if (!successor->left)
                                        break;
                                temp = successor;
                        }
                        temp->left = successor->right;
                        successor->left = current->left;
                        successor->right = current->right;
                        if (current == root) {
                                root = successor;
                                free(current);
                                return;
                        }
                        (flag > 0) ? (parent->left = successor) :
                                        (parent->right = successor);
                }
        }
        free (current);
        return;
}
```

7.3 Time and Space Complexity

The deletion function requires time proportional to the height(h) of the tree in the worst case that is O(logn) . It takes time to execute O(log n) in the average case and O(n) time in the worst case (in case of skewed binary search tree).

The space complexity for deleting a word from the tree is O(1).

CHAPTER 8 : Total Words in Lexicon

For counting the total number of words in our English Lexicon, the function TotalWords is created and implemented. It is a recursive function. It traverses the binary search tree formed and returns the total number of nodes present.

- TotalWords(root)
  - if Root == NULL:
    - Return;
  - total_words = 1
  - total_words += Total_Words(root->left);
  - Total_words += Total_Words(root->right);
  - return total_words

8.1 Algorithm

- The function first checks if the root is Null or not.
- If the root is not NULL, it counts the word at the root hence initializing total words equal to 1
- It then recursively calls for counting total words for the left child as root and right child as root and add the results from these two recursive calls in total words and finally returns the total words.

8.2 Pseudo Code

```
unsigned int TotalWords(struct BSTnode *root)
{
/*This function counts the total number of words in the lexicon. It is a recursive function.*/

    //no word node present
    if(root==NULL){
        return 0;
    }
    unsigned int count = 1; //one for root node
    if(root->left != NULL){
        count += TotalWords(root->left);
    }
    if(root->right != NULL){
    count += TotalWords(root->right);
    }
    return count;
}
```

8.3 Time and Space Complexity

Time Complexity of Total Words function is O(n) in both average and worst case.

The function is a recursive function. Hence a recursive stack will be formed. Therefore the

Space Complexity of Total Words function is O(log(n)) in average case and O(n) in worst case.

CHAPTER 9: Editing an existing word of Lexicon

We call this function to update/edit the existing word in terms of meaning or synonyms or antonym of our Lexicon.

This function asks the user to enter the word that requires editing. It then searches the given word . If the word is found, it gives users the choice to edit the required field and update the field according. And finally it prints the edited node(i.e meaning , antonym and synonym) for the given word.. If the function is not found within the scope of our Lexicon, it print the same and exit from the function.

9.1 Algorithm

void editWord(char * str)

1. Start from the root.
2. If root is null : print (empty lexicon)‘; return
3. Flag = 0 , temp node = root node
4. compare word with root
5. If cmp < 0 : search in left tree (temp = temp->left)
6. If cmp > 0 : search in right tree (temp = temp->right)
7. If cmp = 0 : word found

choice to edit meaning,antonym or synonym using switch case.

b) get new antonym, synonym or meaning and update the same using strcpy.

c) print edited word with its meaning, synonym and antonym

```
cout<<"edited word is ";

cout<<"word: "<<str<<endl;

cout<<"Meaning: "<<temp->meaning<<endl;

cout<<"Synonym: "<<temp->synonym<<endl;
```

cout<<"Antonym: "<<temp->antonym<<endl;

8. Repeat 5 , 6 until condition 7 is reached iteratively
9. If (flag = 0) ;
10. print (word not found) ;
11. return

9.2 Pseudo Code

```
void editWord(char * str)
{

    struct BSTnode *temp =NULL;
int flag =0, res=0;
if(root ==NULL){
cout<<"Lexicon is empty";
return;
}
temp = root;
while(temp){
if((res = strcasecmp(temp->word,str))==0){

        char nMeaning[256];
        char nAntonym[256];
        char nSynonym[256];
        cout<<"what do you want to edit"<<endl;
        cout<<"1. meaning"<<endl;
        cout<<"2. synonym"<<endl;
        cout<<"3. antonym"<<endl;
        cout<<"4. exit"<<endl;

        int ch;
        cin>>ch;

        switch(ch)
        {
            char b[10];
            char c[10];
            char a[10];
        case 1:
            cout<<"Enter new meaning: ";
            cin.getline(a,10);
            cin.getline(nMeaning,256);
            strcpy(temp->meaning,nMeaning);
            break;

        case 2:
            cout<<"Enter new synonym: ";
            cin.getline(b,10);
            cin.getline(nSynonym,256);
           strcpy(temp->synonym,nSynonym);
            break;
```

```
        case 3:
            cout<<"Enter new antonym: ";
            cin.getline(a,10);
            cin.getline(nAntonym,256);
            strcpy(temp->antonym,nAntonym);
            break;

        case 4:

            break;

        }


        cout<<"Edited word is ";
cout<<"word: "<<str<<endl;
cout<<"Meaning: "<<temp->meaning<<endl;
cout<<"Synonym: "<<temp->synonym<<endl;
cout<<"Antonym: "<<temp->antonym<<endl;
flag=1;
break;
}
temp = (res>0) ? temp ->left : temp ->right;
}
if(flag == 0){
cout<<" word not found"<<endl;
}
return;
}
```

9.3 Time and Space Complexity

Time complexity to edit an existing word is :

1. Average case: O(Log n)
2. Worst case: O(n)

Space Complexity to edit an existing word is : O (1)

CHAPTER 10: CODE AND OUTPUT

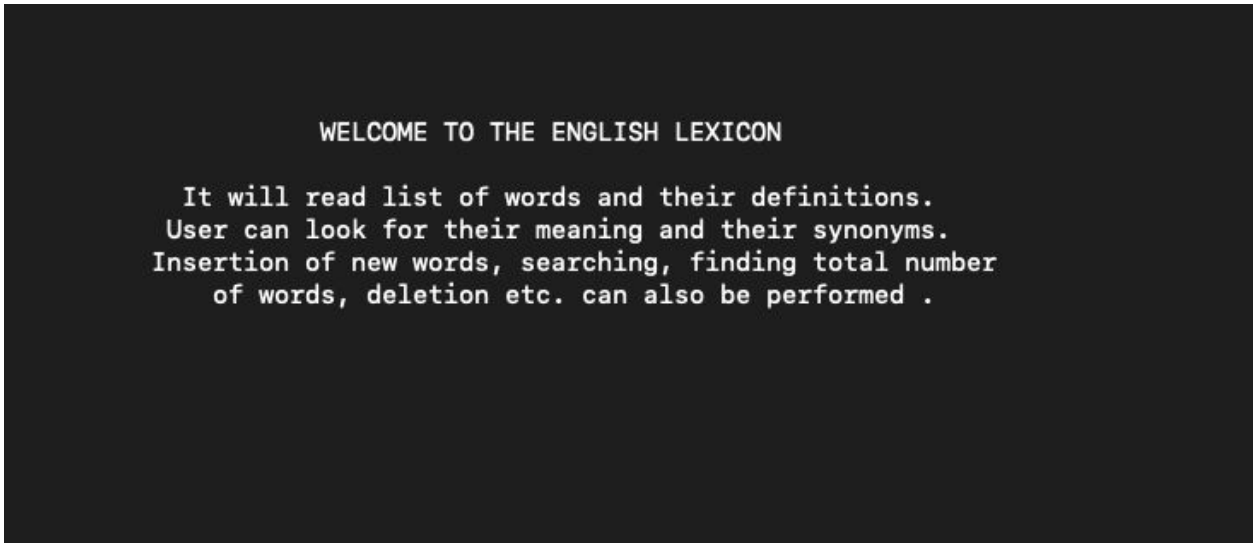
: MAIN FUNCTION :

```
int main ()
{
    char word[256];
    char meaning[256];
    char antonym[256];
    char synonym[256];
    char a[128];
    char b[10];

    bool cond=true;

    cout<<"\n\n\n\n";
    cout<<"\t            WELCOME TO THE ENGLISH LEXICON "<<endl;
    cout<<"\n";
    cout<<"\t   It will read list of words and their definitions."<<endl;
    cout<<"\t  User can look for their meaning and their synonyms."<<endl;
    cout<<"\t Insertion of new words, searching, finding total number"<<endl;
    cout<<"\t     of words, deletion etc. can also be performed ."<<endl;
    cout<<"\n\n\n\n\n\n\n\n";

    while (cond)
    {
        cout<<"-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-"<<endl;
        cout<<"                  : MAIN MENU :"<<endl<<endl;
        cout<<"Enter the number of operation to be performed : "<<endl;
        cout<<"1.Insertion"<<endl;
        cout<<"2.Traversal"<<endl;
        cout<<"3.Search"<<endl;
        cout<<"4.Total words"<<endl;
        cout<<"5.Deletion of a word"<<endl;
        cout<<"6.Updations in a word"<<endl;
        cout<<"7.Exit"<<endl<<endl;
        cout<<"-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-"<<endl;

        int ch;
        cin>>ch;
        switch (ch)
        {
            case 1:
                cout<<" Word to insert: ";
                cin.getline(b,10);
                cin.getline(word,256);
                cout<<" Meaning: ";
                cin.getline(meaning,256);
                cout<<" Synonym: ";
                cin.getline(synonym,256);
                cout<<" Antonym: ";
                cin.getline(antonym,256);
                insert(word,meaning,synonym,antonym);
                cout<<"\n\n\n";
                break;
```

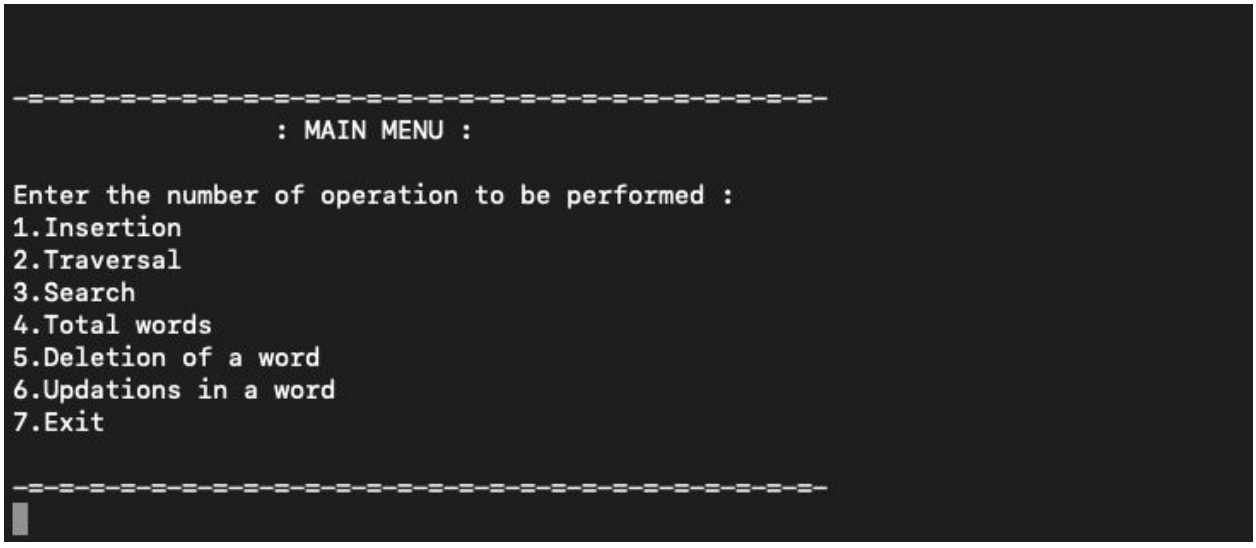
```
            case 2:
                if(root==NULL){
                    cout<<"No words present in the dictionary."<<endl;
                    cout<<"If you'd like to insert some words, press 1."<<endl;
                    break;
                }
                inorderTraversal(root);
                cout<<"\n\n\n";
                break;

            case 3:
                if(root==NULL){
                    cout<<"No words present to search from."<<endl;
                    break;
                }
                cout<<"Word to Search : "<<endl;
                cin.getline(b,10);
                cin.getline(a,128);
                cout<<search(a);
                cout<<"\n\n\n";
                break;

            case 4:
                cout<<"Number of words present at the moment are : "<<TotalWords(root)<<
                cout<<"\n\n\n";
                break;

            case 5:
                cout<<"Enter the word to be deleted from the dictionary : ";
                cin.getline(b,10);
                cin.getline(a,256);
                deleteNode(word);
                cout<<"Requested word has been deleted."<<endl;
                cout<<"\n\n\n";
                break;

            case 6:
                cout<<"enter the word you need to edit: ";
                cin.getline(b,10);
                cin.getline(word,128);
                editWord(word);
                cout<<"\n\n\n";
                break;

            case 7:
                cond=false;
                break;

            default : //re-running the loop
                cout<<"Invalid option chosen."<<endl;
                cout<<"Enter again."<<endl;
        }
    }
    return 0;
```

:OUTPUT SCREENS:

WELCOME PAGE:



Main Menu Page:



Insertion of words to the Lexicon:

```
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
                 : MAIN MENU :

Enter the number of operation to be performed :
1.Insertion
2.Traversal
3.Search
4.Total words
5.Deletion of a word
6.Updations in a word
7.Exit

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
1
 Word to insert: Capable
 Meaning: Having the ability or quality to do or achieve a specified thing.
 Synonym: Competent
 Antonym: Incompetent
```

```
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
                 : MAIN MENU :

Enter the number of operation to be performed :
1.Insertion
2.Traversal
3.Search
4.Total words
5.Deletion of a word
6.Updations in a word
7.Exit

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
1
 Word to insert: Honor
 Meaning: High respect and great esteem.
 Synonym: Adoration
 Antonym: Denunciation
```

```
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
                 : MAIN MENU :

Enter the number of operation to be performed :
1.Insertion
2.Traversal
3.Search
4.Total words
5.Deletion of a word
6.Updations in a word
7.Exit

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
1
 Word to insert: Allure
 Meaning: The quality of being mysteriously attractive.
 Synonym: Fascinate
 Antonym: Repel
```

```
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
                 : MAIN MENU :

Enter the number of operation to be performed :
1.Insertion
2.Traversal
3.Search
4.Total words
5.Deletion of a word
6.Updations in a word
7.Exit

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
1
 Word to insert: Outbreak
 Meaning: A sudden occurence of something unwelcome, such as war or disease.
 Synonym: Insurrection
 Antonym: Subjection
```

Traversal to all the words:

```
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
                : MAIN MENU :

Enter the number of operation to be performed :
1.Insertion
2.Traversal
3.Search
4.Total words
5.Deletion of a word
6.Updations in a word
7.Exit

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
2
Word is Allure
Meaning: The quality of being mysteriously attractive.
Synonym: Repel
Antonym: Fascinate
Word is Capable
Meaning: Having the ability or quality to do or achieve a specified thing.
Synonym: Competent
Antonym: Incompetent
Word is Honor
Meaning: High respect and great esteem.
Synonym: Denunciation
Antonym: Adoration
Word is Outbreak
Meaning: A sudden occurence of something unwelcome, such as war or disease.
Synonym: Subjection
Antonym: Insurrection
```

Searching the word:

```
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
                  : MAIN MENU :

Enter the number of operation to be performed :
1.Insertion
2.Traversal
3.Search
4.Total words
5.Deletion of a word
6.Updations in a word
7.Exit

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
3
Word to Search :
Allure
word:Allure
Meaning:The quality of being mysteriously attractive.
Synonym:Repel
Antonym:Fascinate
```

```
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
                  : MAIN MENU :

Enter the number of operation to be performed :
1.Insertion
2.Traversal
3.Search
4.Total words
5.Deletion of a word
6.Updations in a word
7.Exit

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
3
Word to Search :
Happy
word not found
```

Total number of words:

```
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
                 : MAIN MENU :

Enter the number of operation to be performed :
1.Insertion
2.Traversal
3.Search
4.Total words
5.Deletion of a word
6.Updations in a word
7.Exit

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
4
Number of words present at the moment are : 4
```

Deletion of the word:

```
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
                 : MAIN MENU :

Enter the number of operation to be performed :
1.Insertion
2.Traversal
3.Search
4.Total words
5.Deletion of a word
6.Updations in a word
7.Exit

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
5
Enter the word to be deleted from the dictionary : Allure
Requested word has been deleted.
```

```
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
                 : MAIN MENU :

Enter the number of operation to be performed :
1.Insertion
2.Traversal
3.Search
4.Total words
5.Deletion of a word
6.Updations in a word
7.Exit

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
2
Word is Capable
Meaning: Having the ability or quality to do or achieve a specified thing.
Synonym: Competent
Antonym: Incompetent
Word is Honor
Meaning: High respect and great esteem
Synonym: Denunciation
Antonym: Adoration
Word is Outbreak
Meaning: A sudden occurence of something unwelcome, such as war or disease.
Synonym: Subjection
Antonym: Insurrection
```

Editing the word:

```
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
                  : MAIN MENU :

Enter the number of operation to be performed :
1.Insertion
2.Traversal
3.Search
4.Total words
5.Deletion of a word
6.Updations in a word
7.Exit

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
6
enter the word you need to edit: Honor
what do you want to edit
1. Meaning
2. Synonym
3. Antonym
4. Exit
1
Enter new meaning: The quality of knowing and doing what is morally right.
Edited word is ->
 Word: Honor
 Meaning: The quality of knowing and doing what is morally right.
 Synonym: Denunciation
 Antonym: Adoration
```

Exiting the program :

```
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
                 : MAIN MENU :

Enter the number of operation to be performed :
1.Insertion
2.Traversal
3.Search
4.Total words
5.Deletion of a word
6.Updations in a word
7.Exit

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
7
```

CHAPTER 11: CONCLUSION

Practical knowledge basically means the visualization of the given knowledge, which we study in justour books. For this, we are required to perform experiments, do projects and experiments. Practical knowledge is obviously very important in every field. The main objective of our project was to implement the Lexicon (dictionary) using the principal concepts of trees. Binary search Tree is the most efficient tree in terms of searching for a node, so we've covered it into a coherent framework. The dictionary is a real-life application where a person can look for words, their meanings, along with synonyms and antonyms, easily. We've gained a lot of knowledge and exposure in Data Structures from working on this project, be it concepts, team work, group discussion, innovation, optimization, debugging, testing and development. We also faced a lot of difficulties during the execution of the lexicon, be it errors, be it idea implementation, be it complexity. We'd like to take this project to a level above by adding file handling, etc. in the future.

CHAPTER 12: FUTURE SCOPE

There are a number of directions to enhance the work presented here in this report and to explore new applications which involves trees, and for the ideas about extensibility presented here. Future work concerns deeper analysis of particular mechanisms, efficiency of algorithms in time and space, learning file handling, indexing in trees, using B trees, or B+ trees, or any other efficient m-way trees. New proposals to try different methods, or simply curiosity to develop applications which run on these algorithms. This has a very vast scope in the future.

BIBLIOGRAPHY

[1] Binary Tree Data Structure - GeeksforGeeks

[2]Binary Tree and its Types | Data Structure Tutorial | Studytonight

[3]Binary tree - Wikipedia

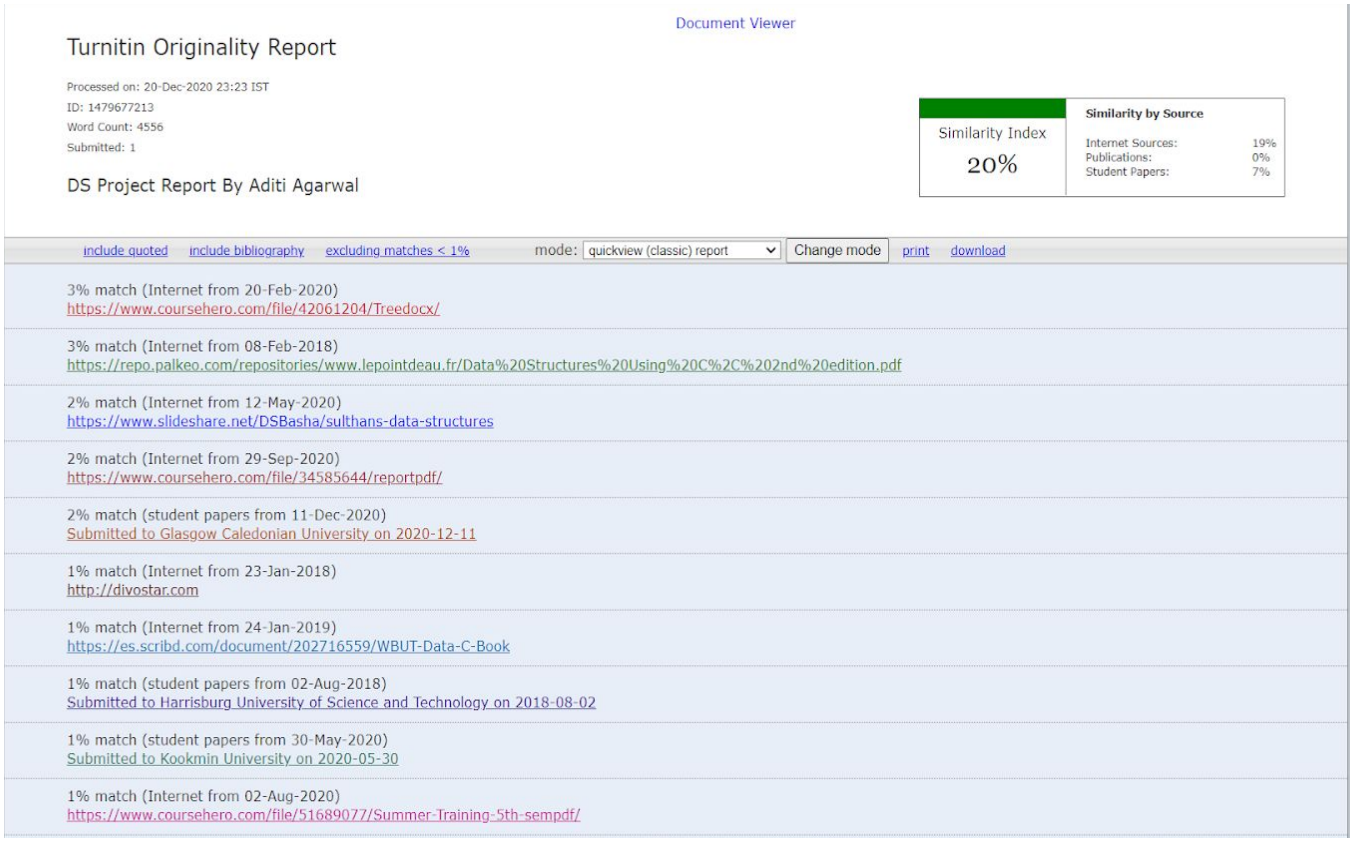
[4]Binary Tree - javatpoint

[5]Data Structures using C - Tenenbaum

[6]Data Structure and Algorithms - Tree - Tutorialspoint

PLAGIARISM REPORT

The Turnitin Plagiarism Report can be found at:
https://drive.google.com/file/d/1nn2JqPqU4kGDlhJoRhsNSXE1WQzw0pdD/view?usp=sharing



Document Viewer

Turnitin Originality Report

Processed on: 20-Dec-2020 23:23 IST
ID: 1479677213
Word Count: 4556
Submitted: 1

DS Project Report By Aditi Agarwal

Similarity Index 20%

Similarity by Source
Internet Sources: 19%
Publications: 0%
Student Papers: 7%

include quoted | include bibliography | excluding matches < 1% | mode: quickview (classic) report | Change mode | print | download

3% match (Internet from 20-Feb-2020)
https://www.coursehero.com/file/42061204/Treedocx/

3% match (Internet from 08-Feb-2018)
https://repo.palkeo.com/repositories/www.lepointdeau.fr/Data%20Structures%20Using%20C%2C%202nd%20edition.pdf

2% match (Internet from 12-May-2020)
https://www.slideshare.net/DSBasha/sulthans-data-structures

2% match (Internet from 29-Sep-2020)
https://www.coursehero.com/file/34585644/reportpdf/

2% match (student papers from 11-Dec-2020)
Submitted to Glasgow Caledonian University on 2020-12-11

1% match (Internet from 23-Jan-2018)
http://divostar.com

1% match (Internet from 24-Jan-2019)
https://es.scribd.com/document/202716559/WBUT-Data-C-Book

1% match (student papers from 02-Aug-2018)
Submitted to Harrisburg University of Science and Technology on 2018-08-02

1% match (student papers from 30-May-2020)
Submitted to Kookmin University on 2020-05-30

1% match (Internet from 02-Aug-2020)
https://www.coursehero.com/file/51689077/Summer-Training-5th-sempdf/