|  |  |
| --- | --- |
| **VISVESVARAYA TECHNOLOGICAL UNIVERSITY**  JNANA SANGAMA, BELAGAVI – 590 018    **A Data Structure (18CS32) Mini Project Report on**  **IMPLEMENTATION OF DICTIONARY USING BINARY SEARCH TREE**  *Submitted in partial fulfillment of the requirements as a part of the Data structure*  *Lab for theI II Semester of degree of* ***Bachelor of Engineering in Information***  ***Science and Engineering*** *of Visvesvaraya Technological University, Belagavi*    **Submitted by** | |
|  | **PHALGUNI S PRASAD PRAGNA G PRASAD PRAJNA S G PRAJWAL B P BARLAYA**  **1RN20IS105**  **1RN20IS106 1RN20IS107 1RN20IS108** |
| **Under the Guidance of**  **Faculty Incharge**  **Mrs. VINUTHA G K**  **Assistant Professor**  **Dept. of ISE, RNSIT**            **Department of Information Science and Engineering**  **RNS Institute of Technology**  Channasandra, Dr. Vishnuvardhan Road, RR Nagar Post,  Bengaluru – 560 098    **2021 – 2022** |

**Contents**

**Concepts Page NO**

1.Introduction 3

2.Data Structure 4

3.Flow Chart / Algorithm 5

4.Enhancements 6

5.Code for the System 7

**Introduction**

# 1.Overview

# Implementation of Dictionary using BST

* Dictionary can be implemented using binary search tree.
* A binary search tree is a binary tree such that each node stores a key of a dictionary.
* Key 'k' of a node is always greater than the keys present in its left sub tree.
* Similarly, key 'k' of a node is always lesser than the keys present in its right sub tree.

# 2.Objective

* To implement a simple dictionary ADT using a binary search tree.
* Word dictionary is one of the materials frequently used by all people. As we all know it is used to find the meaning of a desired word.
* This is an effort to recreate the dictionary using trees concept.

**3. Description (Functions)**

1. *INSERT WORD ( )*

This function is used to create a new BST node and assign values to it by taking word and its meaning.

**b)** *SEARCH WORD ( )*

This function is used to search meaning of the word by entering the word.

1. *TRAVERSAL ( )*

This function is used to access all nodes and print the word and its meaning.

1. *DELETE WORD/NODE ( )*

This function is used to delete the specific word from dictionary.

**Data Structure Used:**

**TREE**  **a non-linear data structure which organizes data in a hierarchical manner.**

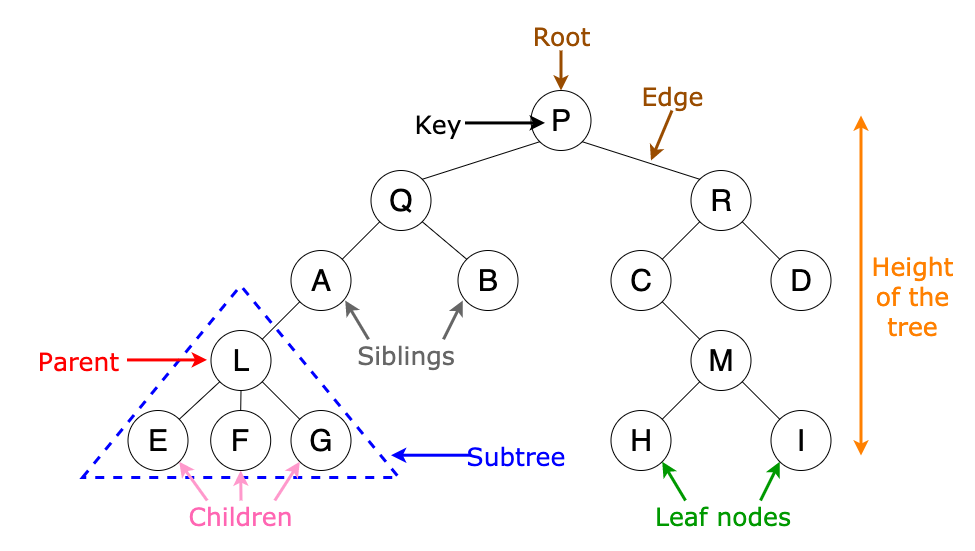
**It is a hierarchical structure as elements in a Tree are arranged in multiple levels.**

A tree is a collection of nodes. Each Node contains 2 fields , data field and link field. Nodes are connected by edges.

* **A tree has the following properties**:

1. The tree has one node called root. The tree originates from this node, and hence it does not have any parent.
2. Each node has only one parent but a parent node can have many child nodes ,also called sub-trees.
3. The parent node and the child node is connected through an edge.

* **The various terminologies in trees / representation of tree:**



**Flow Chart / Algorithm :**

* Insert a node

1. Create a new BST node and assign values to it.
2. insert(node, key)

  i) if root == NULL,

         return the new node to the calling function.

     ii) if root=>data < key

         call the insert function with root=>right and assign the return value in root=>right.

         root->right = insert(root=>right,key)

     iii) if root=>data > key

         call the insert function with root->left and assign the return value in root=>left.

         root=>left = insert(root=>left,key)

3. Finally, return the original root pointer to the calling function

* Delete Word

1.Input the data of the node to be deleted.

2. If root is NULL BST is empty and stop.

3. If data < root->item then

Call delete() in left subtree and stop.

4. If data > root->item then

Call delete() in right subtree and stop.

5. If the node is a leaf node, delete the node directly and stop.

6. If the node has one child, copy the child to the node to be deleted and delete the child node and stop.

7. If the node has two children

I. Copy the contents of the inorder successor to to be deleted

II. Delete the inorder successor and stop.

* Search word

1. If root -> data = item or root = null. Return ROOT.
2. Else. If root < root -> data. Return search(root -> LEFT, ITEM)
3. ELSE. Return search(root -> RIGHT,ITEM)
4. [END OF IF] [END OF IF]

## Future Enhancements of the system

1.File concept can be included to permanently store the wordsand store new set of words everytime the user inputs the word.

2.Number of fields in each node can be increased to hold 2 or more meanings of the same word.

3.Front end application can be built to enhance the user experience.

**CODE for IMPLEMENTATION OF DICTIONARY USING BST**

**Pre-processor Directives**

The # commands in C/C++ are called the pre-processor directives. They are a set of instructions to the "pre-processor" to make changes to your code BEFORE compilation.

\*\*\*\*\*\*\*\*\*\*\*CODE \*\*\*\*\*\*\*\*\*\*\*

#include <stdio.h>

#include <stdlib.h>

#include<string.h>

## Structure declaration

A Structure is a key word that create use defined data type in C language. A structure creates data type that can be used to group items of possibly different types into a single type.

‘struct’ keyword is used to create structure.

A structure variable can either be declared with structure declaration or as a separate declaration like basic types.

\*\*\*\*\*\*\*\*\*\*\*CODE \*\*\*\*\*\*\*\*\*\*\*

## struct BSTnode {         char word[128], meaning[256];         struct BSTnode \*left, \*right;

## INSERT WORD ( )

void insert(char \*word, char \*meaning)

{ struct BSTnode \*parent = NULL, \*current = NULL, \*newnode = NULL;

int res = 0;

if (!root)

{

root = createNode(word, meaning);

return;

}

for (current = root; current !=NULL;

current = (res > 0) ? current->right : current->left)

{

res = strcasecmp(word, current->word);

if (res == 0)

{

printf("Duplicate entry!!\n");

return;

}

parent = current;

}

newnode = createNode(word, meaning);

res > 0 ? (parent->right = newnode) : (parent->left = newnode);

return;

}

**DELETE WORD ( )**

void deleteNode(char \*str)

{

struct BSTnode \*parent = NULL, \*current = NULL, \*temp = NULL;

int flag = 0, res = 0;

if (!root)

{

printf("BST is not present!!\n");

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;

}

if (current->right == NULL) /\* deleting leaf node \*/

{

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;

}

## 

## SEARCH WORD( )

void findElement(char \*str)

{

struct BSTnode \*temp = NULL;

int flag = 0, res = 0;

if (root == NULL)

{

printf("Binary Search Tree is out of station!!\n");

return;

}

temp = root;

while (temp)

{

if ((res = strcasecmp(temp->word, str)) == 0)

{

printf("Word : %s", str);

printf("Meaning: %s", temp->meaning);

flag = 1;

break;

}

temp = (res > 0) ? temp->left : temp->right;

}

if (!flag)

printf("Search Element not found in Binary Search Tree\n");

return;

}

**TRAVERSAL ( )**

void inorderTraversal(struct BSTnode \*myNode)

{

if (myNode)

{

inorderTraversal(myNode->left);

printf("Word : %s", myNode->word);

printf("Meaning : %s", myNode->meaning);

printf("\n");

inorderTraversal(myNode->right);

}

return;

}

## Int main( )

Main function tells the compiler where the program starts. The main function in this program firstly shows the all options the user has while running the program and based on the choice selected by the user the main function calls the required function using the switch statement. Returns an integer.

\*\*\*\*\*\*\*\*\*\*\*CODE \*\*\*\*\*\*\*\*\*\*\*

int main() {

        int ch;

        char str[128], meaning[256];

        while (1) {

                printf("\n1. Insertion\t2. Deletion\n");

                printf("3. Searching\t4. Traversal\n");

                printf("5. Exit\nEnter ur choice:");

                scanf("%d", &ch);

                getchar();

                switch (ch) {

                        case 1:

                                printf("Word to insert:");

                                fgets(str, 100, stdin);

                                printf("Meaning:");

                                fgets(meaning, 256, stdin);

                                insert(str, meaning);

                                break;

                        case 2:

                                printf("Enter the word to delete:");

                                fgets(str, 100, stdin);

                                deleteNode(str);

                                break;

                        case 3:

                                printf("Enter the search word:");

                                fgets(str, 100, stdin);

                                findElement(str);

                                break;

                        case 4:

                                inorderTraversal(root);

                                break;

                        case 5:

                                exit(0);

                        default:

                                printf("You have entered wrong option\n");

                                break;

                }

        }

        return 0;

  }

***THANK YOU***