Vivekanand Education Society's
Institute of Technology
Approved by AICTE & Affiliated to University of Mumbai

COMPUTER ENGINEERING

DS ODD SEM 2021-22/EXPERIMENT 9

NAME:- GAURAV AMARNANI (D7A, 67)

Experiment - 9

AIM: Application of Binary Search Technique

Theory:

A binary search tree also known as an ordered binary tree is a variant of binary tree in which the nodes are arranged in an order. A binary tree is a non-linear data structure which is a collection of elements called nodes. In a binary tree the topmost element is called the root-node. An element can have 0, 1 at the most 2 child ~~notes~~ nodes.

The order of binary tree is

- All the values in the left sub-tree has a value less than that of a root node.
- All the values in the right node has a greater value than the value of a root node.
- The same rule is carried forward to all the subtree in tree.

Since the tree is already ordered the time taken to carry out a search operation on the tree is generally reduced as now we don't have to traverse the entire tree but at every sub-tree we get hint where to search next. Binary trees also help in speed up the insertion and deletion operation.

Algorithm to insert element into Binary Search Tree

Step 1: IF TREE NULL
Allocate memory for TREE SET TREE->DATA=ITEM
SET TREE -> LEFT = TREE -> RIGHT NULL

ELSE
IF ITEM < TREE -> DATA
INSERT TREE -> LEFT , ITEM)
ELSE
INSERT (TREE -> RIGHT , ITEM)
[END of IF)
(END of IF)

Step 2: END

Algorithm to search element in binary tree

Step 1: IF ROOT -> DATA = ITEM OR
ROOT = NULL
RETURN ROOT
ELSE
IF ROOT < ROOT -> DATA
RETURN SEARCH (ROOT -> RIGHT ITEM)

[END of IF]
[END of IF]

Step-2: END

Algorithm to delete an element from binary tree

Step 1: IF TREE = NULL
Write item not found in tree ELSE IF ITEM <
TREE → DATA Delete (TREE → LEFT, ITEM)
ELSE IF ITEM → TREE → DATA. Delete (TREE →
RIGHT, ITEM) ELSE IF TREE → LEFT AND
TREE → RIGHT
SET TEMP = find Largest Node (TREE → LEFT)
SET TREE → DATA = TEMP → DATA
Delete (TREE → LEFT, TEMP → DATA)
ELSE
SET TEMP = TREE
IF TREE → LEFT = NULL AND TREE → RIGHT = NULL
SET TREE = NULL
ELSE IF TREE → LEFT = NULL
SET TREE = TREE → LEFT
ELSE
SET TREE = TREE → RIGHT
[END of IF]
FREE TEMP
[END of IF]

Step 2: END

Algorithm for inorder traversal for binary search tree

Step 1 : Repeat Steps 2 to 4 While TREE = NULL
Step 2 : INORDER (TREE -> LEFT)
Step 3 : Write TREE -> DATA
Step 4 : INORDER (TREE -> RIGHT)
[END OF LOOP]
Step 5 : END

Algorithm for preorder traversal for binary search tree

Step 1: Repeat Steps 2 to 4 While TREE = NULL
Step 2: Write TREE -> DATA
Step 3: PREORDER (TREE -> LEFT)
Step 4 : PREORDER (TREE -> RIGHT)
[END of LOOP]
Step 5 : END

Algorithm for Postorder Traversal in binary search tree

Step 1 : Repeat Steps 2 to 4 While TREE ! = NULL
Step 2 : POSTORDER (TREE -> LEFT)
Step 3 : POSTORDER (TREE -> RIGHT)
Step 4: Write TREE -> DATA
[END of LOOP]
Step 5 : END

Algorithm to count total number of nodes in binary tree (Internal)

Step 1: IF TREE = NULL
Return 0
IF TREE → LEFT = NULL AND TREE → RIGHT = NULL
Return 0
ELSE
Return Total Internal Nodes (Tree)
total Internal nodes (TREE → RIGHT)
[END OF IF]

Step 2: END

Algorithm to count total number of nodes in binary Search tree (External)

Step 1: IF TREE = NULL
RETURN 0
ELSE IF TREE → LEFT = NULL AND TREE → RIGHT = NULL
Return 1
ELSE
Return Total External Node (TREE → LEFT) +
total External Node (TREE → RIGHT)
[END OF IF]

Step 2: END

Algorithm to Determine Height of a Binary Search Tree

Step 1: IF there TREE = NULL
Return 0
ELSE
SET LEFT HEIGHT = HEIGHT (TREE → LEFT)
SET RIGHT HEIGHT = HEIGHT (TREE → RIGHT)
IF LEFT HEIGHT > RIGHT HEIGHT
Return LEFT HEIGHT + 1
ELSE
Return RIGHT HEIGHT + 1

Step 2: END

Algorithm to obtain mirror image of binary Search tree

Step 1: IF TREE != NULL
MIRROR IMAGE (TREE → LEFT)
MIRROR IMAGE (TREE → RIGHT)
SET TEMP = TREE → LEFT
SET TREE → LEFT = TREE → RIGHT
SET TREE → RIGHT = TEMP
[END of IF]

Step 2: END

Algorithm to find smallest node in binary search tree

Step 1 : IF TREE = NULL OR TREE → LEFT = NULL
RETURN TREE
ELSE
RETURN SMALEST ELEMENT (TREE → LEFT)
[END of IF]

Step 2 : END

Algorithm to find largest node in binary search tree

Step 1: IF TREE → NULL OR TREE → RIGHT = NULL
RETURN TREE
ELSE
RETURN FIND LARGEST ELEMENT (TREE → RIGHT)
[END OF IF]

Step 2: END

Conclusion:

Hence by performing this experiment we learned various application and techniques of Binary Search Tree and also implemented different techniques into code.

Program:

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
struct node {
int data;
struct node *left; struct node *right;
};
struct node *tree;
void create_tree(struct node *);
struct node *insertElement(struct node *, int);
void preorderTraversal(struct node *);
void inorderTraversal(struct node *);
void postorderTraversal(struct node *);
struct node *findSmallestElement(struct node *);
struct node *findLargestElement(struct node *);
struct node *deleteElement(struct node *, int);
struct node *mirrorImage(struct node *);
struct node *deleteTree(struct node *);
int totalNodes(struct node *);
int totalExternalNodes(struct node *);
int totalInternalNodes(struct node *);
int Height(struct node *);
int main() {
int option, val;
struct node *ptr;
create_tree(tree);
do {
printf("\n ***MAIN MENU** \n");
printf("\n 1. Insert Element");
printf("\n 2. Preorder Traversal");
printf("\n 3. Inorder Traversal");
printf("\n 4. Postorder Traversal");
printf("\n 5. Find the smallest element");
printf("\n 6. Find the largest element");
printf("\n 7. Delete an element");
printf("\n 8. Count the total number of nodes");
printf("\n 9. Determine the height of the tree");
printf("\n 10. Find the mirror image of the tree");
printf("\n 11. Delete the tree");
printf("\n 12. Exit");
printf("\n\n Enter your option : ");
scanf("%d", &option);
switch (option) {
case 1:
printf("\n Enter the value of the new node : ");
scanf("%d", &val);
tree = insertElement(tree, val);
break;
case 2:
printf("\n The elements of the tree are : \n");
preorderTraversal(tree);
break;
case 3:
printf("\n The elements of the tree are : \n");
inorderTraversal(tree);
break;
case 4:
printf("\n The elements of the tree are : \n");
postorderTraversal(tree);
```

```
break;
case 5:
ptr = findSmallestElement(tree);
printf("\n Smallest element is :%d", ptr->data);
break;
case 6:
ptr = findLargestElement(tree);
printf("\n Largest element is : %d", ptr->data);
break;
case 7:
printf("\n Enter the element to be deleted : ");
scanf("%d", &val);
tree = deleteElement(tree, val);
break;
case 8:
printf("\n Total no. of nodes = %d", totalNodes(tree));
break;
case 9:
printf("\n The height of the tree = %d", Height(tree));
break;
case 10:
tree = mirrorImage(tree);
break; case 11:
tree = deleteTree(tree);
break;
}
} while (option != 14);
getch();
return 0;
}
void create_tree(struct node *tree) {
tree = NULL;
}
struct node *insertElement(struct node *tree, int val) {
struct node *ptr, *nodeptr, *parentptr;
ptr = (struct node *)malloc(sizeof(struct node));
ptr->data = val;
ptr->left = NULL; ptr->right = NULL;
if (tree == NULL) {
tree = ptr;
tree->left = NULL;
tree->right = NULL;
}
else {
parentptr = NULL;
nodeptr = tree;
while (nodeptr != NULL) {
parentptr = nodeptr;
if (val< nodeptr->data) nodeptr = nodeptr->left;
else
}
nodeptr = nodeptr->right;
if (val<parentptr->data) parentptr->left = ptr;
else
}
parentptr->right = ptr;
return tree;
}
void preorderTraversal(struct node *tree) {
if (tree != NULL) {
printf("%d\t", tree->data);
preorderTraversal(tree->left);
```

```
preorderTraversal(tree->right);
}
}
void inorderTraversal(struct node *tree) {
if (tree != NULL) {
inorderTraversal(tree->left);
printf("%d\t", tree->data);
inorderTraversal(tree->right);
}
}
void postorderTraversal(struct node *tree) {
if (tree != NULL) {
postorderTraversal(tree->left);
postorderTraversal(tree->right);
printf("%d\t", tree->data);
}
}
struct node *findSmallestElement(struct node *tree) {
if ((tree == NULL) || (tree->left == NULL)) return tree;
else
return findSmallestElement(tree->left);
}
struct node *findLargestElement(struct node *tree) {
if ((tree == NULL) || (tree->right == NULL))
return tree;
else
}
return findLargestElement(tree->right);
struct node *deleteElement(struct node *tree, int val) {
struct node *cur, *parent, *suc, *psuc, *ptr;
if (tree->left == NULL) {
printf("\n The tree is empty ");
return (tree);
}
parent = tree; cur = tree->left;
while (cur != NULL && val != cur->data) {
parent = cur;
cur = (val< cur->data) ? cur->data : cur->right;
}
if (cur == NULL) {
printf("\n The value to be deleted is not present in the tree");
return (tree);
}
if (cur->left == NULL) ptr = cur->right;
else if (cur->right == NULL) ptr = cur->left;
else {
psuc = cur;
cur = cur->left;
while (suc->left != NULL) {
psuc = suc;
suc = suc->left;
}
if (cur == psuc) {
suc->left = cur->right;
}
suc->left = cur->right;
psuc->left = suc->right;
suc->right = cur->right;
ptr = suc;
}
parent->left = ptr;
else
```

```
parent->right = ptr;
free(cur);
return tree;
}
int totalNodes(struct node *tree) {
if (tree == NULL)
return 0;
else
}
return (totalNodes(tree->left) + totalNodes(tree->right) + 1);
int Height(struct node *tree) {
int leftheight, rightheight;
if (tree == NULL)
return 0;
else {
leftheight = Height(tree->left);
rightheight = Height(tree->right);
if (leftheight > rightheight)
return (leftheight + 1);
else
}
}
return (rightheight + 1);
struct node *mirrorImage(struct node *tree) {
struct node *ptr;
if (tree != NULL) {
mirrorImage(tree->left);
mirrorImage(tree->right);
ptr = tree->left;
ptr->left = ptr->right;
tree->right = ptr;
}
}
struct node *deleteTree(struct node *tree) {
if (tree != NULL) {
deleteTree(tree->left);
deleteTree(tree->right);
free(tree);
}
}
```

Output:

```
     9. Determine the height of the tree
     10. Find the mirror image of the tree
     11. Delete the tree
     12. Exit

     Enter your option : 1

     Enter the value of the new node : 50

     ***MAIN MENU**

     1. Insert Element
     2. Preorder Traversal
     3. Inorder Traversal
     4. Postorder Traversal
     5. Find the smallest element
     6. Find the largest element
     7. Delete an element
     8. Count the total number of nodes
     9. Determine the height of the tree
     10. Find the mirror image of the tree
     11. Delete the tree
     12. Exit

     Enter your option :
```

```
 9. Determine the height of the tree
 10. Find the mirror image of the tree
 11. Delete the tree
 12. Exit

 Enter your option : 2

 The elements of the tree are :
10      20      30      40      50
 ***MAIN MENU**

 1. Insert Element
 2. Preorder Traversal
 3. Inorder Traversal
 4. Postorder Traversal
 5. Find the smallest element
 6. Find the largest element
 7. Delete an element
 8. Count the total number of nodes
 9. Determine the height of the tree
 10. Find the mirror image of the tree
 11. Delete the tree
 12. Exit

 Enter your option : _
```

```
 9. Determine the height of the tree
 10. Find the mirror image of the tree
 11. Delete the tree
 12. Exit

 Enter your option : 3

 The elements of the tree are :
10      20      30      40      50
 ***MAIN MENU**

 1. Insert Element
 2. Preorder Traversal
 3. Inorder Traversal
 4. Postorder Traversal
 5. Find the smallest element
 6. Find the largest element
 7. Delete an element
 8. Count the total number of nodes
 9. Determine the height of the tree
 10. Find the mirror image of the tree
 11. Delete the tree
 12. Exit

 Enter your option : _
```

```
9. Determine the height of the tree
10. Find the mirror image of the tree
11. Delete the tree
12. Exit

Enter your option : 4

The elements of the tree are :
50      40      30      20      10
***MAIN MENU**

1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Find the smallest element
6. Find the largest element
7. Delete an element
8. Count the total number of nodes
9. Determine the height of the tree
10. Find the mirror image of the tree
11. Delete the tree
12. Exit

Enter your option :
```

```
8. Count the total number of nodes
9. Determine the height of the tree
10. Find the mirror image of the tree
11. Delete the tree
12. Exit

Enter your option : 5

Smallest element is :10
***MAIN MENU**

1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Find the smallest element
6. Find the largest element
7. Delete an element
8. Count the total number of nodes
9. Determine the height of the tree
10. Find the mirror image of the tree
11. Delete the tree
12. Exit

Enter your option : _
```

```
8. Count the total number of nodes
9. Determine the height of the tree
10. Find the mirror image of the tree
11. Delete the tree
12. Exit

Enter your option : 6

Largest element is : 50
***MAIN MENU**

1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Find the smallest element
6. Find the largest element
7. Delete an element
8. Count the total number of nodes
9. Determine the height of the tree
10. Find the mirror image of the tree
11. Delete the tree
12. Exit

Enter your option : _
```

```
10. Find the mirror image of the tree
11. Delete the tree
12. Exit

Enter your option : 7

Enter the element to be deleted : 10

The tree is empty
***MAIN MENU**

1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Find the smallest element
6. Find the largest element
7. Delete an element
8. Count the total number of nodes
9. Determine the height of the tree
10. Find the mirror image of the tree
11. Delete the tree
12. Exit

Enter your option : _
```

```
8. Count the total number of nodes
9. Determine the height of the tree
10. Find the mirror image of the tree
11. Delete the tree
12. Exit

Enter your option : 8

Total no. of nodes = 5
***MAIN MENU**

1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Find the smallest element
6. Find the largest element
7. Delete an element
8. Count the total number of nodes
9. Determine the height of the tree
10. Find the mirror image of the tree
11. Delete the tree
12. Exit

Enter your option : _
```

```
8. Count the total number of nodes
9. Determine the height of the tree
10. Find the mirror image of the tree
11. Delete the tree
12. Exit

Enter your option : 9

The height of the tree = 5
***MAIN MENU**

1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Find the smallest element
6. Find the largest element
7. Delete an element
8. Count the total number of nodes
9. Determine the height of the tree
10. Find the mirror image of the tree
11. Delete the tree
12. Exit

Enter your option : _
```