**EXPERIMENT NO. 07**

|  |  |
| --- | --- |
| **DATE OF PERFORMANCE:** | **GRADE:** |
| **DATE OF ASSESSMENT:** | **SIGNATURE OF LECTURER/ TTA:** |
|  |  |

**AIM: IMPLEMENTATION OF TREE.**

**THEORY:**

**TREE:** Tree represents nodes connected by edges. **Binary Tree** is a special data structure used for data storage purposes. A binary tree is made of nodes, where each node contains a "left" reference, a "right" reference, and a data element. The topmost node in the tree is called the root. A binary tree has a special condition that each node can have two children at maximum. 

**BST:** A BST is a binary tree where nodes are ordered in the following way:

* each node contains one key (also known as data)
* the keys in the left sub tree are less than the key in its parent node
* the keys in the right sub tree are greater than the key in its parent node
* Duplicate keys are not allowed.

**OPERATION ON BINARY TREE:**

**INSERTION:** insert an element in a tree / create a tree.The very first insertion creates the tree. Afterwards, whenever an element is to be inserted. First locate its proper location. Start search from root node then if data is less than key value, search empty location in left subtree and insert the data. Otherwise search empty location in right subtree and insert the data.

**ALGORITHM:**

If root is NULL

then create root node

return

If root exists then

compare the data with node.data

while until insertion position is located

If data is greater than node.data

goto right subtree

else

goto left subtree

endwhile

insert data

end If

**DELETION:** Deletion of node has three cases:

* Removing a leaf (no children): removing a leaf is simply a matter of setting the appropriate link of its parent to null.
* Removing a node with only one child: make the reference from the parent skip over the removed node and point instead to the child of the node we intend to remove.
* Removing a node with two children: replaces the node’s info with the info from another node in the tree so that the search property is retained - then remove this other node.

**TRAVERSAL:** Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot random access a node in tree. There are three ways which we use to traverse a tree −

**PREORDER:** In this traversal method, the root node is visited first, then left sub tree and finally right sub-tree.

**ALGORITHM:**

(Until all nodes are traversed)

Step 1 − Visit root node.

Step 2 − Recursively traverse left sub tree.

Step 3 − Recursively traverse right sub tree.

**INORDER:** In this traversal method, the left left-subtree is visited first, then root and then the right sub-tree. We should always remember that every node may represent a sub tree itself.

**ALGORITHM:**

(Until all nodes are traversed)

Step 1 − Recursively traverse left sub tree.

Step 2 − Visit root node.

Step 3 − Recursively traverse right sub tree.

**POSTORDER:** In this traversal method, the root node is visited last, hence the name. First we traverse left sub tree, then right sub tree and finally root.

**ALGORITHM:**

(Until all nodes are traversed)

Step 1 − Recursively traverse left sub tree.

Step 2 − Recursively traverse right sub tree.

Step 3 − Visit root node.

**SEARCH:** Whenever an element is to be search. Start search from root node then if data is less than key value, search element in left sub tree otherwise search element in right sub tree.

**ALGORITHM:**

If root.data is equal to search.data

return root

else

while data not found

If data is greater than node.data

goto right subtree

else

goto left subtree

If data found

return node

endwhile

return data not found

end if

**PROGRAM 1: IMPLEMENTATION OF TREE.**

**#include<stdio.h>**

**#include<alloc.h>**

**#include<conio.h>**

**#include<stdio.h>**

**struct tree**

**{**

**int info;**

**struct tree \*left;**

**struct tree \*right;**

**};**

**struct tree \*insert(struct tree \*,int);**

**void inorder(struct tree \*);**

**void postorder(struct tree \*);**

**void preorder(struct tree \*);**

**struct tree \*delet(struct tree \*,int);**

**struct tree \*search(struct tree \*);**

**int main(void)**

**{**

**struct tree \*root;**

**int choice, item,item\_no;**

**root = NULL;**

**clrscr();**

**/\* rear = NULL;\*/**

**do**

**{**

**do**

**{**

**printf("\n \t 1. Insert in Binary Tree ");**

**printf("\n\t 2. Delete from Binary Tree ");**

**printf("\n\t 3. Inorder traversal of Binary tree");**

**printf("\n\t 4. Postorder traversal of Binary tree");**

**printf("\n\t 5. Preorder traversal of Binary tree");**

**printf("\n\t 6. Search and replace ");**

**printf("\n\t 7. Exit ");**

**printf("\n\t Enter choice : ");**

**scanf(" %d",&choice);**

**if(choice<1 || choice>7)**

**printf("\n Invalid choice - try again");**

**}while (choice<1 || choice>7);**

**switch(choice)**

**{**

**case 1:**

**printf("\n Enter new element: ");**

**scanf("%d", &item);**

**root= insert(root,item);**

**printf("\n root is %d",root->info);**

**printf("\n Inorder traversal of binary tree is : ");**

**inorder(root);**

**break;**

**case 2:**

**printf("\n Enter the element to be deleted : ");**

**scanf(" %d",&item\_no);**

**root=delet(root,item\_no);**

**inorder(root);**

**break;**

**case 3:**

**printf("\n Inorder traversal of binary tree is : ");**

**inorder(root);**

**break;**

**case 4:**

**printf("\n Postorder traversal of binary tree is : ");**

**postorder(root);**

**break;**

**case 5:**

**printf("\n Preorder traversal of binary tree is : ");**

**preorder(root);**

**break;**

**case 6:**

**printf("\n Search and replace operation in binary tree ");**

**root=search(root);**

**break;**

**default:**

**printf("\n End of program ");**

**} /\* end of switch \*/**

**}while(choice !=7);**

**return(0);**

**}**

**struct tree \*insert(struct tree \*root, int x)**

**{**

**if(!root)**

**{**

**root=(struct tree\*)malloc(sizeof(struct tree));**

**root->info = x;**

**root->left = NULL;**

**root->right = NULL;**

**return(root);**

**}**

**if(root->info > x)**

**root->left = insert(root->left,x);**

**else**

**{**

**if(root->info < x)**

**root->right = insert(root->right,x);**

**}**

**return(root);**

**}**

**void inorder(struct tree \*root)**

**{**

**if(root != NULL)**

**{**

**inorder(root->left);**

**printf(" %d",root->info);**

**inorder(root->right);**

**}**

**return;**

**}**

**void postorder(struct tree \*root)**

**{**

**if(root != NULL)**

**{**

**postorder(root->left);**

**postorder(root->right);**

**printf(" %d",root->info);**

**}**

**return;**

**}**

**void preorder(struct tree \*root)**

**{**

**if(root != NULL)**

**{**

**printf(" %d",root->info);**

**preorder(root->left);**

**preorder(root->right);**

**}**

**return;**

**}**

**struct tree \*delet(struct tree \*ptr,int x)**

**{**

**struct tree \*p1,\*p2;**

**if(!ptr)**

**{**

**printf("\n Node not found ");**

**return(ptr);**

**}**

**else**

**{**

**if(ptr->info < x)**

**{**

**ptr->right = delet(ptr->right,x);**

**/\*return(ptr);\*/**

**}**

**else if (ptr->info >x)**

**{**

**ptr->left=delet(ptr->left,x);**

**return ptr;**

**}**

**else /\* no. 2 else \*/**

**{**

**if(ptr->info == x) /\* no. 2 if \*/**

**{**

**if(ptr->left == ptr->right) /\*i.e., a leaf node\*/**

**{**

**free(ptr);**

**return(NULL);**

**}**

**else if(ptr->left==NULL) /\* a right subtree \*/**

**{**

**p1=ptr->right;**

**free(ptr);**

**return p1;**

**}**

**else if(ptr->right==NULL) /\* a left subtree \*/**

**{**

**p1=ptr->left;**

**free(ptr);**

**return p1;**

**}**

**else**

**{**

**p1=ptr->right;**

**p2=ptr->right;**

**while(p1->left != NULL)**

**p1=p1->left;**

**p1->left=ptr->left;**

**free(ptr);**

**return p2;**

**}**

**}/\*end of no. 2 if \*/**

**}/\* end of no. 2 else \*/**

**/\* check which path to search for a given no. \*/**

**}**

**return(ptr);**

**}**

**/\* function to search and replace an element in the binary tree \*/**

**struct tree \*search(struct tree \*root)**

**{**

**int no,i,ino;**

**struct tree \*ptr;**

**ptr=root;**

**printf("\n Enter the element to be searched :");**

**scanf(" %d",&no);**

**fflush(stdin);**

**while(ptr)**

**{**

**if(no>ptr->info)**

**ptr=ptr->right;**

**else if(no<ptr->info)**

**ptr=ptr->left;**

**else**

**break;**

**}**

**if(ptr)**

**{**

**printf("\n Element %d which was searched is found and is = %d",no,ptr->info);**

**printf("\n Do you want replace it, press 1 for yes : ");**

**scanf(" %d",&i);**

**if(i==1)**

**{**

**printf("\n Enter new element :");**

**scanf(" %d",&ino);**

**ptr->info=ino;**

**}**

**else**

**printf("\n\t It's okay");**

**}**

**else**

**printf("\n Element %d does not exist in the binary tree",no);**

**return(root);**

**}**

**OUTPUT:**