Experiment 21
Binary Search Tree

Date: 31-12-2020

Aim: Implement a Binary Search Tree

Data Structures used: Linked List, Binary Tree

Algorithm for Insertion

Input: The root node (root) and the key, element to be inserted
Output : The binary search tree with the node inserted
Data Structure : Binary Search Tree

Steps

1. Step 1: Start
2. Step 2: ptr = root
3. Step 3: while(ptr!=NULL and flag==true) do
4. Step 1: case: item<=ptr → data
5. Step 1: ptr1 = ptr
6. Step 2: ptr=ptr→ lc
7. Step 2: case: item>ptr → data
8. Step 1: ptr1=ptr
9. Step 2: ptr = ptr→ rc
10. Step 3: endCase
11. Step 4: endWhile
12. Step 5: if(ptr==NULL) then
13. Step 1: new = getNode(node)
14. Step 2: new → data = item
15. Step 3: new→ rc = new→lc = NULL
16. Step 4: if(ptr→ dara <= item) then
17. Step 1: ptr1→ rc = new
18. Step 5: else
19. Step 1: ptr1→lc = new
20. Step 6: endIf
21. Step 6: endif
22. Step 7: Stop

Algorithm for Deleting a node

Input: Root node of the binary search tree, the element to be deleted
Output: Binary tree with the element deleted
Data Structure used: Binary search tree

Steps

1. Step 1: Start
2. Step 2: ptr = root
3. Step 3: flage = false
4. Step 4: while(ptr!=NULL) then
5. Step 1: case: item < ptr→ data
6. Step 1: parent = ptr
7. Step 2: ptr = ptr→lc
8. Step 2: case item > ptr→data
9. Step 1: parent = ptr

10. Step 2: ptr = ptr→rc
11. Step 3: case item=ptr→data
12. Step 1: flage = true
13. Step 4: endcase
14. Step 5: endWhile
15. Step 6: if(flag = false) then
16. Step 1: printf("There is no item in the binary tree")
17. Step 2: exit
18. Step 7: endIf
19. Step 8: If(ptr→lc==NULL and ptr→ rc ==NULL) then //case 1
20. Step 1: if(parent → lc == ptr) then
21. Step 1: parent→lc =NULL
22. Step 2: else
23. Step 1: parent →rc =NULL
24. Step 3: endIf
25. Step 4: returnNode(ptr)
26. Step 9: else if(ptr→lc !=NULL and ptr-.rc !=NULL) then //case 3
27. Step 1: ptr1 = ptr →rc
28. Step 2: while(ptr1→lc!=NULL) do
29. Step 1: ptr1= ptr1 → lc
30. Step 3: endWhile
31. Step 4: item = ptr1→data
32. Step 5: delete_node(ptr1)
33. Step 6: ptr → data = item
34. Step 10: else //case 2
35. Step 1: if(parent→lc == ptr) then
36. Step 1: if (ptr→lc ==NULL) then
37. Step 1: parent →lc = ptr→rc
38. Step 2: else
39. Step 1: parent →lc = ptr → lc
40. Step 3: endIf
41. Step 2: else
42. Step 1: if(ptr → lc ==NULL) then
43. Step 1: parent →rc = ptr→rc
44. Step 2: else
45. Step 1: parent → rc = ptr → lc
46. Step 3: endif
47. Step 3: EndIf
48. Step 11: endif
49. Step 12: Stop

Algorithm for Inorder Traversal

Input: Root node of the binary tree
Output : All the nodes of the binary tree visited in an inorder fashion
Data Structure used: Binary trees

Steps

1. Step 1: Start
2. Step 2: if(root!=NULL) then
3. Step 1: inorder_traversal(root→lc)
4. Step 2: visit(root)
5. Step 3: inorder_traversal(root→ rc)
6. Step 3: else
7. Step 1: return
8. Step 4: endif
9. Step 5: Stop

Algorithm for Postorder Traversal

Input: Root node of the binary tree
Output : All the nodes of the binary tree visited in an postorder fashion
Data Structure used: Binary trees

Steps

10. Step 1: Start
11. Step 2: if(root!=NULL) then
12. Step 1: postorder_traversal(root→lc)
13. Step 2: postorder_traversal(root→ rc)
14. Step 3: visit(root)
15. Step 3: else
16. Step 1: return
17. Step 4: endif
18. Step 5: Stop

Algorithm for Preorder Traversal

Input: Root node of the binary tree
Output : All the nodes of the binary tree visited in an preorder fashion
Data Structure used: Binary trees

Steps

19. Step 1: Start
20. Step 2: if(root!=NULL) then
21. Step 1: visit(root)
22. Step 2: preorder_traversal(root→lc)
23. Step 3: preorder_traversal(root→ rc)
24.
25. Step 3: else
26. Step 1: return
27. Step 4: endif
28. Step 5: Stop

Program Code

```
/**********************
 * Binary Search Tree
 * Done By Rohit Karunakaran
 * ********************/


#include<stdio.h>
#include<stdlib.h>

typedef struct binary_search_tree_node{
    struct binary_search_tree_node* lc;
    struct binary_search_tree_node* rc;
    int value;
}node;

node* search_node(node* root, int value){
    if(root!=NULL){
        if(root->value!=value){
            if(root->value>value){
```

```
                return search_node(root->lc,value);
            }
            else{
                return search_node(root->rc,value);
            }
        }
        else{
            return root;
        }
    }
    else{
        return NULL;
    }
}


void insert_node(node** root,int value){
    int flag=1;
    node* ptr=*root;
    if(ptr!=NULL){
        while(ptr!=NULL&&flag){
            if(ptr->value<value){
                if(ptr->rc==NULL){
                    ptr->rc = (node*)malloc(sizeof(node));
                    ptr->rc->lc = ptr->rc->rc =NULL;
                    ptr->rc->value = value;
                    flag=0;
                }
                else{
                    ptr= ptr->rc;
                }
            }
            else{
                if(ptr->lc==NULL){
                    ptr->lc = (node*)malloc(sizeof(node));
                    ptr->lc->lc = ptr->lc->rc =NULL;
                    ptr->lc->value = value;
                    flag=0;
                }
                else{
                    ptr = ptr->lc;
                }
            }
        }
    }
    else{
    //Root is empty
        *root = (node*)malloc(sizeof(node));
        (*root) ->lc = (*root)->rc = NULL;
        (*root)->value = value;
    }
}

void delete_node(node** root, int value,node* par){
    node* ptr = *root;
    node* parent =par;
```

```
int flag = 1;
if(ptr!=NULL){
    while(ptr!=NULL&&flag){
        if(ptr->value<value){
            parent = ptr;
            ptr = ptr->rc;
        }
        else if(ptr->value>value){
            parent = ptr;
            ptr = ptr->lc;
        }
        else{
            flag = 0;
        }
    }
    if(flag == 1){
        printf("Item not found\n");
        return;
    }
    if(ptr ->lc ==NULL && ptr->rc==NULL){
        if(parent!=NULL){
            if(parent -> rc ==ptr){
                parent ->rc =NULL;
            }
            else {
                parent ->lc =NULL;
            }
        }
        else{
            *root = NULL;
        }
        free(ptr);

    }
    else if(ptr->lc!=NULL && ptr->rc!=NULL){
        node* ptr1=ptr->rc;
        while(ptr1->lc!=NULL) ptr1=ptr1->lc; //Find the successor node
        int item = ptr1->value;
        delete_node(&ptr1,item,ptr);
        ptr->value = item;
    }
    else{
        if(parent!=NULL){
            if(parent ->rc ==ptr){
               if(ptr->rc!=NULL){
                    parent ->rc = ptr->rc;
               }
               else{
                    parent->rc = ptr->lc;
               }
            }
            else{
               if(ptr->rc!=NULL){
                    parent ->lc = ptr->rc;
               }
               else{
```

```
                        parent->lc = ptr->lc;
                    }

                }
            }
            else{
                //If the parent is null then the node is root and has one child
                if(ptr->rc!=NULL){
                    *root = ptr->rc;
                }
                else{
                    *root = ptr->lc;
                }
            }
            free(ptr);
        }
    }
    else{
        printf("There is no item in the binary tree\n");
    }
}

void inorder_traversal(node* root){
    if(root!=NULL){
        inorder_traversal(root->lc);
        printf("%d ",root->value);
        inorder_traversal(root->rc);

    }
    else{
        return;
    }

}

void postorder_traversal(node* root){
    if(root!=NULL){
        postorder_traversal(root->lc);
        postorder_traversal(root->rc);
        printf("%d ",root->value);
    }
    else{
        return;
    }
}

void preorder_traversal(node* root){
    if(root!=NULL){
        printf("%d ",root->value);
        preorder_traversal(root->lc);
        preorder_traversal(root->rc);
    }
    else{
        return;
    }
}
```

```
void leaf_nodes(node* root,int* count){
    if(root!=NULL){
        leaf_nodes(root->lc,count);
        if(root->lc==NULL&&root->rc==NULL)(*count)++;
        leaf_nodes(root->rc,count);

    }
    else{
        return;
    }

}


int menu(node* root){
    printf("Binary Tree implementation\n");
    int RUN=1;
    int choice;
    int elem;
    while(RUN){
        printf("\nMenu\n");
        printf("1.Insert\n");
        printf("2.Inorder traversal\n");
        printf("3.Preorder traversal\n");
        printf("4.Postorder traversal\n");
        printf("5.Delete Node\n");
        printf("6.Number of leaf nodes\n");
        printf("7. Exit\n");
        printf("Enter Choice: ");
        scanf("%d",&choice);
        switch(choice){
            case 1: printf("Enter the value to be inserted : ");
                    scanf("%d",&elem);
                    insert_node(&root,elem);
                    break;
            case 2: if(root!=NULL){
                        printf("Inorder Traversal: ");
                        inorder_traversal(root);
                    }
                    else
                        printf("The tree is Empty!!!!\n");
                    break;
            case 3: if(root!=NULL){
                        printf("Preorder Traversal: ");
                        preorder_traversal(root);
                    }
                    else
                        printf("The tree is Empty!!!!\n");
                    break;
            case 4: if(root!=NULL){
                        printf("Postorder Traversal: ");
                        postorder_traversal(root);
                    }
                    else
                        printf("The tree is Empty!!!!\n");
```

```
                break;
            case 5: printf("Enter the value to be deleted: ");
                scanf("%d",&elem);
                delete_node(&root,elem,NULL);
                break;

            case 6: if(root!=NULL){
                    elem = 0;
                    leaf_nodes(root,&elem);
                    printf("Number of leafnodes = %d\n",elem)
                }
                else{
                    printf("The tree is empty there is no leaf nodes\n");
                }
            case 7: RUN=0;
                break;
            default:printf("Wrong value entered try again\n\n");
                break;
        }

    }

    return RUN;
}

int main(){
    node* root = NULL;
    return menu(root);
}
```

Result: The program compiled successfully and required output was obtained

Sample input and output

```
..ograming/C/CSL201/2020-12-31❯ ./binarySearchTree.o
Binary Tree implementation

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 1
Enter the value to be inserted : 12

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 1
Enter the value to be inserted : 11

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 1
Enter the value to be inserted : 14

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 1
Enter the value to be inserted : 35
```

```
Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 1
Enter the value to be inserted : 24

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 2
Inorder Traversal: 11 12 14 24 35
Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 3
Preorder Traversal: 12 11 14 35 24
Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 4
Postorder Traversal: 11 24 35 14 12
```

```
Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 5
Enter the value to be deleted: 12

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 5
Enter the value to be deleted: 11

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 2
Inorder Traversal: 14 24 35
Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 5
Enter the value to be deleted: 24
```

```
Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 6
Number of leafnodes = 1

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 7
..ograming/C/CSL201/2020-12-31❯
```