Date: 10.10.2023
Ex. No: 6

Name: Niranjan.B
Reg. No.: 3122225001082

UCS 2312 Data Structures Lab

Assignment 5: BSTADT and its application

Date of Assignment: 20.10.2023

Create an ADT for the binary search tree data structure with the following functions. Each node which consists of integer data, address of left and right children.

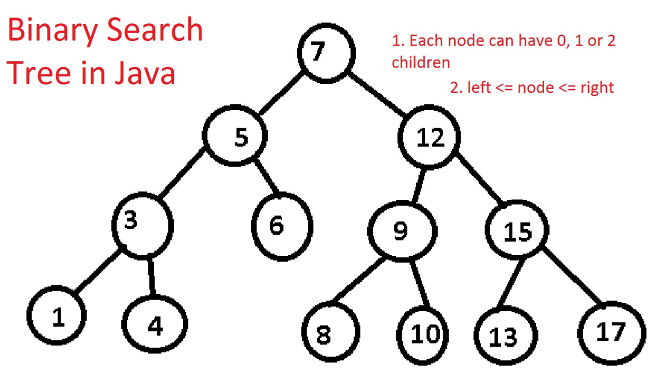
[CO2, K3]

a. insertBST(t,data) – insert data into BST
b. inorder(t) – display the tree using inorder traversal
c. preorder(t) – display the tree using preorder traversal
d. postorder(t) – display the tree using postorder traversal
e. levelorder(t) – display the tree hierarchically
f. findmin(t)– returns the minimum element in the tree
g. search(t,key) – returns the element found, otherwise returns NULL
h. delete(t,elt) – delete the given elt from tree

Write an application to do the following

a. Check whether the two BST contains the same set of elements
b. Count the number of nodes in tree within the given range
c. Find sum of k smallest elements in the given BST

Data Structure – Binary Search Tree:




```
struct tree
{
    int data;
    struct tree *left,*right;
};
```

Algorithm –

Algorithm: Insert data into BST

Input – Pointer to tree, data to be added to tree

Output – struct tree *

1. if (t==NULL)
   t=(struct tree *)malloc(sizeof(struct tree));
   t->data=data;
   t->right=NULL;
   t->left=NULL;
2. if(data>t->data)
   t->right=insert(t->right,data)
3. if(data<t->data)
   t->left=insert(t->left,data)
4. return t

Algorithm: Inorder

Input – Pointer to tree

Output – void

1. if (t==NULL)
   return
2. if(t->left!=NULL)
   inorder(t->left)
3. print data in t
4. if(t->right!=NULL)
   inorder(t->right)

Algorithm: Preorder

Input – Pointer to tree

Output – void

1. if (t==NULL)
   return
2. print data in t
3. if(t->left!=NULL)
   inorder(t->left)
4. if(t->right!=NULL)
   inorder(t->right)

Algorithm: Postorder

Input – Pointer to tree

Output – void

1. if (t==NULL)
   return
2. if(t->left!=NULL)
   inorder(t->left)
3. if(t->right!=NULL)
   inorder(t->right)
4. print data in t

Algorithm: Levelorder
Input – Pointer to tree, level of node
Output – void
1. if (t==NULL)
        return
2. if(l==1)
        print data in t
3. if(l>1)
        level(t->left,l-1)
        level(t->right,l-1)

Algorithm: Returns the minimum element in the tree
Input – Pointer to tree
Output – struct tree *
1. if (t->left==NULL)
        return t
2. findmin(t->left)

Algorithm: Returns the element found, otherwise returns NULL
Input – Pointer to tree, data to be found
Output – struct tree *
1. if (t==NULL || t->data==key)
        return t
2. if(key<t->data)
        return search(t->left,key)
3. if(key>t->data)
        return search(t->right,key)

Algorithm: Delete the given elt from tree
Input – Pointer to tree, data to be deleted
Output – struct tree *
1. if (data<t->data)
        t->left=delete(t->left,data)
2. else if (data>t->data)
        t->left=delete(t->right,data)
3. else if(t->left && t->right)
        temp=findmin(t->right);
        t->data=temp->data;
        t->right=delete(t->right,temp->data);
4. else
        temp=t;
        if(t->right==NULL)
                t=t->left;
        else if(t->left==NULL)
                t=t->right;
        free(temp);
5. return t

Name: Niranjan.B
Reg. No.: 3122225001082

tree.h code:

```
struct node
{
	int data;
	struct node * next;
};

void append(struct node* header,int data)
{
	struct node* temp;
	temp=(struct node *)malloc(sizeof(struct node));
	temp->data=data;
	struct node *ptr,*end;
	ptr=header->next;
	end=header;
	while(ptr!=NULL)
	{
		end=ptr;
		ptr=ptr->next;
	}
	temp->next=end->next;
	end->next=temp;
}

struct tree
{
	int data;
	struct tree *left,*right;
};

struct tree * insert(struct tree *t, int data)
{
	if(t==NULL)
	{
		t=(struct tree *)malloc(sizeof(struct tree));
		t->data=data;
		t->right=NULL;
		t->left=NULL;
	}
	if(data>t->data)
	{
		t->right=insert(t->right,data);
	}
	if(data<t->data)
	{
		t->left=insert(t->left,data);
	}
	return t;
}

void inorder(struct tree *t)
{
	if (t==NULL)
		return;
```

```
	if(t->left!=NULL)
	{
		inorder(t->left);
	}
	printf(" %d",t->data);
	if(t->right!=NULL)
	{
		inorder(t->right);
	}
}

void inorder1(struct tree *t,struct node *h)
{
	if (t==NULL)
		return;
	if(t->left!=NULL)
	{
		inorder1(t->left,h);
	}
	append(h,t->data);
	if(t->right!=NULL)
	{
		inorder1(t->right,h);
	}
}

void postorder(struct tree *t)
{
	if (t==NULL)
		return;
	if(t->left!=NULL)
	{
		postorder(t->left);
	}
	if(t->right!=NULL)
	{
		postorder(t->right);
	}
	printf(" %d",t->data);
}

void preorder(struct tree *t)
{
	if (t==NULL)
		return;
	printf(" %d",t->data);
	if(t->left!=NULL)
	{
		preorder(t->left);
	}
	if(t->right!=NULL)
	{
		preorder(t->right);
	}
```

```
}

struct tree * findmin(struct tree *t)
{
	if(t->left==NULL)
	{
		return t;
	}
	findmin(t->left);
}

struct tree * delete(struct tree *t,int data)
{
	struct tree *temp;
	if(data<t->data)
	{
		t->left=delete(t->left,data);
	}
	else if(data>t->data)
	{
		t->right=delete(t->right,data);
	}
	else if(t->left && t->right)
	{
		temp=findmin(t->right);
		t->data=temp->data;
		t->right=delete(t->right,temp->data);
	}
	else
	{
		temp=t;
		if(t->right==NULL)
		{
			t=t->left;
		}
		else if(t->left==NULL)
		{
			t=t->right;
		}
		free(temp);
	}
	return t;
}

struct tree* search(struct tree* t, int key)
{
  if(t==NULL || t->data==key)
      return t;
  if(key<t->data)
      return search(t->left,key);
  if(key>t->data)
      return search(t->right,key);
}
```

```
int height(struct tree* t)
{
    if (t == NULL)
        return 0;
    else
    {
        int lheight = height(t->left);
        int rheight = height(t->right);
        if (lheight > rheight)
            return (lheight + 1);
        else
            return (rheight + 1);
    }
}

void level (struct tree* t, int l)
{
  if (t==NULL)
      return;
  if (l==1)
      printf("%d ", t->data);
  else if (l>1)
  {
      level(t->left, l-1);
      level(t->right, l-1);
  }
}
```

tree.c code:

```
#include<stdio.h>
#include<stdlib.h>
#include "tree.h"
void main()
{
      struct tree *t=NULL,*s;
      int choice =1,data,key;
      while(choice)
      {

      printf("\n\n1.Insert\n2.Inorder\n3.Preorder\n4.Postorder\n5.Delete\
n6.Search\n7.Level Order\n0.Exit\nChoice : ");
            scanf("%d",&choice);
            switch(choice)
            {
                  case 1:
                  {
                        printf("Data = ");
                        scanf("%d",&data);
                        t=insert(t,data);
                        break;
                  }
                  case 2:
                  {
                        printf("Inorder t:");
                        inorder(t);
```

```
                        break;
                }
                case 3:
                {
                        printf("Preorder t:");
                        preorder(t);
                        break;
                }
                case 4:
                {
                        printf("Postorder t:");
                        postorder(t);
                        break;
                }
                case 5:
                {
                        printf("Data to be deleted = ");
                        scanf("%d",&data);
                        t=delete(t,data);
                        break;
                }
                case 6:
                {
                printf ("Enter element to search: ");
                scanf ("%d", &key);
                s = search (t,key);
                if (s==NULL)
                        printf ("Element not found.");
                else
                        printf ("Element %d found.", s->data);
                break;
                }
                case 7:
                {
                for (int i=0; i<=height (t); i++)
                {
                        level(t,i);
                        printf("\n");
                }
                break;
                }
                default:printf("Invalid Choice");
        }
    }
}
```

Output Screen:

Insert(t,29)
Insert(t,23)
Insert(t,4)
Insert(t,13)
Insert(t,39)
Insert(t,31)
Insert(t,45)
Insert(t,56)
Insert(t,49)

```
1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Delete
6.Search
7.Level Order
0.Exit
Choice : 2
Inorder t: 4 13 23 29 31 39 45 49 56

1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Delete
6.Search
7.Level Order
0.Exit
Choice : 3
Preorder t: 4 13 23 29 31 39 45 49 56

1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Delete
6.Search
7.Level Order
0.Exit
Choice : 4
Postorder t: 56 49 45 39 31 29 23 13 4
```

```
1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Delete
6.Search
7.Level Order
0.Exit
Choice : 2
Inorder t: 4 13 23 29 31 39 45 49 56

1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Delete
6.Search
7.Level Order
0.Exit
Choice : 5
Data to be deleted = 4


1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Delete
6.Search
7.Level Order
0.Exit
Choice : 2
Inorder t: 13 23 29 31 39 45 49 56

1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Delete
6.Search
7.Level Order
0.Exit
Choice : 5
Data to be deleted = 39


1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Delete
6.Search
7.Level Order
0.Exit
Choice : 2
Inorder t: 13 23 29 31 45 49 56
```

```
1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Delete
6.Search
7.Level Order
0.Exit
Choice : 6
Enter element to search: 29
Element 29 found.

1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Delete
6.Search
7.Level Order
0.Exit
Choice : 6
Enter element to search: 4
Element not found.
```

```
1.Insert
2.Inorder
3.Preorder
4.Postorder
5.Delete
6.Search
7.Level Order
0.Exit
Choice : 7

4
13
23
29
31
39
45
49
56
```

Name: Niranjan.B
Reg. No.: 3122225001082

APPLICATIONS:

a. **Check whether the two BST contains the same set of elements**

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include "tree.h"
int compare(struct tree *t1,struct tree *t2)
{
         struct node *h1,*h2;
    h1=(struct node *)malloc(sizeof(struct node));
         h1->next=NULL;
    h2=(struct node *)malloc(sizeof(struct node));
         h2->next=NULL;
         inorder1(t1,h1);
         inorder1(t2,h2);
         struct node *ptr1,*ptr2;
         ptr1=h1->next;
         ptr2=h2->next;
         while(ptr1!=NULL && ptr2!=NULL)
         {
           if(ptr1->data!=ptr2->data)
                return 0;
           ptr1=ptr1->next;
           ptr2=ptr2->next;
         }
    if(ptr1!=NULL || ptr2!=NULL)
           return 0;
         return 1;
}
void main()
{
         struct tree *t1=NULL;
         struct tree *t2=NULL;
         int choice =1,data;
         while(choice)
         {
           printf("\n\n1.Insert t1\n2.Insert
t2\n3.Inorder\n4.Preorder\n5.Postorder\n6.Delete t1\n7.Delete
t2\n8.Compare\n0.Exit\nChoice : ");
           scanf("%d",&choice);
           switch(choice)
           {
                case 1:
                {
                      printf("Data = ");
                      scanf("%d",&data);
                      t1=insert(t1,data);
                      break;
                }
```

```
case 2:
{
      printf("Data = ");
      scanf("%d",&data);
      t2=insert(t2,data);
      break;
}
case 3:
{
      printf("Inorder t1:");
      inorder(t1);
      printf("\nInorder t2:");
      inorder(t2);
      break;
}
case 4:
{
      printf("Preorder t1:");
      preorder(t1);
      printf("\nPreorder t2:");
      preorder(t2);
      break;
}
case 5:
{
      printf("Postorder t1:");
      postorder(t1);
      printf("\nPostorder t2:");
      postorder(t2);
      break;
}
case 6:
{
      printf("Data to be deleted = ");
      scanf("%d",&data);
      t1=delete(t1,data);
      break;
}
case 7:
{
      printf("Data to be deleted = ");
      scanf("%d",&data);
      t2=delete(t2,data);
      break;
}
case 8:
{
      if(compare(t1,t2))
            printf("Same tree");
```

```
                    else
                        printf("Different tree");
                    break;
                }
                default:printf("Invalid Choice");
            }
        }
}
```

Output:

Tree1- insert(t,2), insert(t,1), insert(t,4), insert(t,6)

Tree2- insert(t,1), insert(t,2), insert(t,4), insert(t,6)

```
1.Insert t1
2.Insert t2
3.Inorder
4.Preorder
5.Postorder
6.Delete t1
7.Delete t2
8.Compare
0.Exit
Choice : 3
Inorder t1: 1 2 4 6
Inorder t2: 1 2 4 6

1.Insert t1
2.Insert t2
3.Inorder
4.Preorder
5.Postorder
6.Delete t1
7.Delete t2
8.Compare
0.Exit
Choice : 4
Preorder t1: 2 1 4 6
Preorder t2: 1 2 4 6

1.Insert t1
2.Insert t2
3.Inorder
4.Preorder
5.Postorder
6.Delete t1
7.Delete t2
8.Compare
0.Exit
Choice : 8
Same tree
```

b. Count the number of nodes in tree within the given range

Code:

```
int count_nodes (struct tree* t, int start, int end, int c)
{
  if (t->data>=start && t->data<=end)
    c++;
  else if (t==NULL)
    return c;
  else
  {
    if (t->left!=NULL)
    count_nodes (t->left,start,end,c);
    if (t->right!=NULL)
    count_nodes (t->right,start,end,c);
  }
}
```

Output:

Tree- insert(t,2), insert(t,4), insert(t,6), insert(t,1)

Range: 1-3

```
No. of nodes in the range 1-3: 2
```

c. Find sum of k smallest elements in the given BST

Code:

```
int sum (struct tree* t, int k, int a[])
{
  int sum=0;
  array (t,a,0);
  for (int i=0;i<k;i++)
    sum+=a[i];
  return sum;
}
```

Output:

Tree- insert(t,2), insert(t,4), insert(t,6), insert(t,1)

k=2

```
Sum of 2 smallest elements: 3
```

Name: Niranjan.B
Reg. No.: 3122225001082

Learning Outcome:

Learning Outcome		
Design	3	Understood design of binary tree
Understanding of DS	3	Understood its applications
Use of DS	3	and its operations
Debugging	3	Was able to fix errors
Best Practices		
Design before coding	3	Designed Properly.
Usage of algorithmic notation	2	Can be improved
Use of multi file C program	3	Used multiple files
Versioning of code	3	Versioned Properly