Algorithm for Binary Search Tree Insertion

To insert a new value *v into a binary search tree T , we use the procedure TREEINSERT.*

The procedure is passed a node *z for which key[z] = v, left[z] = NIL,* and *right[z] = NIL. It modifies T and some of the fields of z in such a way that z* is inserted into an appropriate position in the tree.

TREE-INSERT*(T, z)*

1 *y ← NIL*

2 *x ← root[T ]*

3 **while *x = NIL***

4 **do *y ← x***

5 **if *key[z] < key[x]***

6 **then *x ← left[x]***

7 **else *x ← right[x]***

8 *p[z]← y*

9 **if *y = NIL***

10 **then *root[T ]← z ✄ Tree T was empty***

11 **else if *key[z] < key[y]***

12 **then *left[y]← z***

13 **else *right[y] ← z***

Algorithm for Binary Search Tree Deletion

TREE-DELETE*(T, z)*

1 **if *left[z] = NIL or right[z] = NIL***

2 **then *y ← z***

3 **else *y ← TREE-SUCCESSOR(z)***

4 **if *left[y] = NIL***

5 **then *x ← left[y]***

6 **else *x ← right[y]***

7 **if *x = NIL***

8 **then *p[x] ← p[y]***

9 **if *p[y] = NIL***

10 **then *root[T ] ← x***

11 **else if *y = left[p[y]]***

12 **then *left[p[y]] ← x***

13 **else *right[p[y]] ← x***

14 **if *y = z***

15 **then *key[z]← key[y]***

16 copy *y’s data into z*

17 **return *y***

struct node

{

int data;

struct node \*left;

struct node \*right;

};

struct node \*root = NULL;

void insertNode(struct node \*q, int data)

{

if(q == NULL)

{

struct node \*p =(struct node\*)malloc(sizeof(struct node));

p->left = NULL;

p->right = NULL;

p->data = data;

root = p;

return;

}

else if(data < q->data && q->left != NULL)

{

q = q->left;

insertNode(q,data);

}

else if(data > q->data && q->right != NULL)

{

q = q->right;

insertNode(q,data);

}

else if(q->left == NULL && data < q->data)

{

struct node \*p = (struct node\*)malloc(sizeof(struct node));

p->data = data;

p->right = NULL;

p->left = NULL;

q->left = p;

return;

}

else if(q->right == NULL && data > q->data)

{

struct node \*p = (struct node\*)malloc(sizeof(struct node));

p->data = data;

p->right = NULL;

p->left = NULL;

q->right = p;

return;

}

}

void preOrderTrav(struct node \*q) {

if(q != NULL) {

printf("%d\n",q->data);

preOrderTrav(q->left);

preOrderTrav(q->right);

}

}

main()

{

int n;

int num\_nodes = 7;

for(int x=0;x<num\_nodes;x++)

{

printf("Enter new node of BST ");

scanf("%d",&n);

insertNode(root,n);

}

printf("\nInorder Traversal is:\n");

inOrderTrav(root);

printf("\nPreorder Traversal is:\n");

preOrderTrav(root);

printf("\nPostorder Traversal is:\n");

postOrderTrav(root);

}