Ex. No. 7 AVL Trees

Date:

**Aim:**

To perform insertion operation on an AVL tree and to maintain balance factor.

**Algorithm:**

1. Start

2. Perform standard BST insert for w

3. Starting from w, travel up and find the first unbalanced node. Let z be the first unbalanced node, y be the child of z that comes on the path from w to z and x be the grandchild of z that comes on the path from w to z.

4. Re-balance the tree by performing appropriate rotations on the subtree rooted with z. There can be 4 possible cases that needs to be handled as x, y and z can be arranged in 4 ways.

a) y is left child of z and x is left child of y (Left Left Case)

b) y is left child of z and x is right child of y (Left Right Case)

c) y is right child of z and x is right child of y (Right Right Case) d) y is right child of z and x is left child of y (Right Left Case)

5. Stop

**Program:**

/\* AVL Tree \*/

#include <stdio.h>

#include <stdlib.h>

#define CHANGED 0

#define BALANCED 1

typedef struct bnode

{

int data,bfactor;

struct bnode \*left;

struct bnode \*right;

}node;

int height;

void displaymenu()

{

printf("\nBasic Operations in AVL tree");

printf("\n0.Display menu list");

printf("\n1.Insert a node in AVL tree");

printf("\n2.View AVL tree");

printf("\n3.Exit");

}

node\* getnode()

{

int size;

node \*newnode;

size = sizeof(node);

newnode = (node\*)malloc(size);

return(newnode);

}

void copynode(node \*r, int data)

{

r->data = data;

r->left = NULL;

r->right = NULL;

r->bfactor = 0;

}

void releasenode(node \*p)

{

free(p);

}

node\* searchnode(node \*root, int data)

{

if(root!=NULL)

if(data < root->data)

root = searchnode(root->left, data);

else if(data > root->data)

root = searchnode(root->right, data);

return(root);

}

void lefttoleft(node \*\*pptr, node \*\*aptr)

{

node \*p = \*pptr, \*a = \*aptr;

printf("\nLeft to Left AVL rotation");

p->left = a->right;

a->right = p;

if(a->bfactor == 0)

{

p->bfactor = 1;

a->bfactor = -1;

height = BALANCED;

}

else

{

p->bfactor = 0;

a->bfactor = 0;

}

p = a;

\*pptr = p;

\*aptr = a;

}

void lefttoright(node \*\*pptr, node \*\*aptr, node \*\*bptr)

{

node \*p = \*pptr, \*a = \*aptr, \*b = \*bptr;

printf("\nLeft to Right AVL rotation");

b = a->right;

b->right = p;

if(b->bfactor == 1)

p->bfactor = -1;

else

p->bfactor = 0;

if(b->bfactor == -1)

a->bfactor = 1;

else

a->bfactor = 1;

b->bfactor = 0;

p = b;

\*pptr = p;

\*aptr = a;

\*bptr = b;

}

void righttoright(node \*\*pptr, node \*\*aptr)

{

node \*p = \*pptr, \*a = \*aptr;

printf("\nRight to Right AVL rotation");

p->right = a->left;

a->left = p;

if(a->bfactor == 0)

{

p->bfactor = -1;

a->bfactor = 1;

height = BALANCED;

}

else

{

p->bfactor = 0;

a->bfactor = 0;

}

p = a;

\*pptr = p;

\*aptr = a;

}

void righttoleft(node \*\*pptr, node \*\*aptr, node \*\*bptr)

{

node \*p = \*pptr, \*a = \*aptr, \*b = \*bptr;

printf("\nRight to Left AVL rotation");

b = a->left;

a->left = b->right;

b->right = a;

p->right = b->left;

b->left = p;

if(b->bfactor == -1)

p->bfactor = 1;

else

p->bfactor = 0;

if(b->bfactor == -1)

a->bfactor = 0;

b->bfactor = 0;

p = b;

\*pptr = p;

\*aptr = a;

\*bptr = b;

}

void inorder(node \*root)

{

if(root == NULL)

return;

inorder(root->left);

printf("\n%4d", root->data);

inorder(root->right);

}

void view(node \*root, int level)

{

int k;

if(root == NULL)

return;

view(root->right, level+1);

printf("\n");

for(k=0; k<level; k++)

printf(" ");

printf("%d", root->data);

view(root->left, level+1);

}

node\* insertnode(int data, node \*p)

{

node \*a,\*b;

if(p == NULL)

{

p=getnode();

copynode(p, data);

height = CHANGED;

return(p);

}

if(data < p->data)

{

p->left = insertnode(data, p->left);

if(height == CHANGED)

{

switch(p->bfactor)

{

case -1:

p->bfactor = 0;

height = BALANCED;

break;

case 0:

p->bfactor = 1;

break;

case 1:

a = p->left;

if(a->bfactor == 1)

lefttoleft(&p, &a);

else

lefttoright(&p, &a, &b);

height = BALANCED;

break;

}

}

}

if(data > p->data)

{

p->right = insertnode(data, p->right);

if(height == CHANGED)

{

switch(p->bfactor)

{

case 1:

p->bfactor = 0;

height = BALANCED;

break;

case 0:

p->bfactor = -1;

break;

case -1:

a=p->right;

if(a->bfactor == -1)

righttoright(&p, &a);

else

righttoleft(&p, &a, &b);

height=BALANCED;

break;

}

}

}

return(p);

}

void main()

{

int data, ch;

char choice = 'y';

node \*root = NULL;

system("clear");

displaymenu();

while((choice == 'y') || (choice == 'Y'))

{

printf("\nEnter your choice: ");

fflush(stdin);

scanf("%d",&ch);

switch(ch)

{

case 0:

displaymenu();

break;

case 1:

printf("Enter the value to be inserted ");

scanf("%d", &data);

if(searchnode(root, data) == NULL)

root = insertnode(data, root);

else

printf("\nData already exists");

break;

case 2:

if(root == NULL)

{

printf("\nAVL tree is empty");

continue;

}

printf("\nInorder traversal of AVL tree");

inorder(root);

printf("\nAVL tree is");

view(root, 1);

break;

case 3:

releasenode(root);

exit(0);

}

}

}

**Output:**

**Result :**

Thus rotations were performed as a result of insertions to AVL Tree..